AD-A042 332 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
A MICROPROCESSOR IMPLEMENTATION OF EXTENDED BASIC.(U)
DEC 76 G E EUBANKS

UNCLASSIFIED NL

1 OF 3
ADA042332

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A MICROPROCESSOR IMPLEMENTATION
OF EXTENDED BASIC

by

Gordon Edwin Eubanks, Jr.

December 1976

Thesis Advisor: Gary A. Kildall

Approved for public release; distribution unlimited.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. 3. RECIPIENT'S CATALOG NUMBER (9)
4. TITLE (and Subtitle) (6) A Microprocessor Implementation of Extended Basic,	5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; December 1976
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Gordon Edwin Eubanks, Jr	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940 (11)	12. REPORT DATE December 1976
	13. NUMBER OF PAGES 194
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, California 93940 (12) 194 p.	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.	
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)	
18. SUPPLEMENTARY NOTES	
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) BASIC Extended BASIC Interpreter Compiler Microprocessor	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The design and implementation of an extension to the BASIC programming language for use on a microprocessor-based system has been described. The implementation is comprised of two subsystems, a compiler which generates code for a hypothetical zero-address machine and a run-time monitor which interprets this code. The design goals, solutions, and recommendations for further expansion of the system have been presented. The system

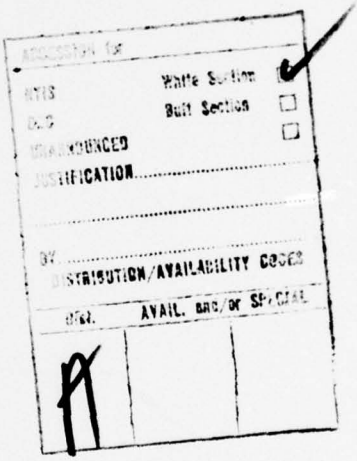
DD FORM 1 JAN 73 1473 (Page 1) EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. (cont.)

was implemented in PL/M for use in a diskette-based environment.



A Microprocessor Implementation
of Extended Basic

by

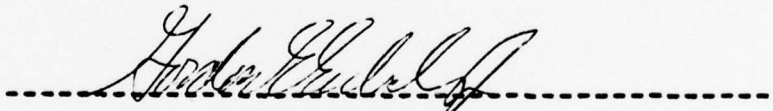
Gordon Edwin Eubanks, Jr.
Lieutenant, United States Navy
B.S., Oklahoma State University, 1968

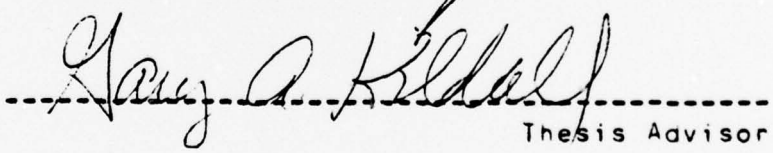
Submitted in partial fulfillment of the
requirements for the degree of

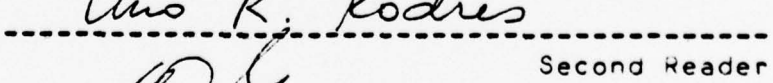
MASTER OF SCIENCE IN COMPUTER SCIENCE

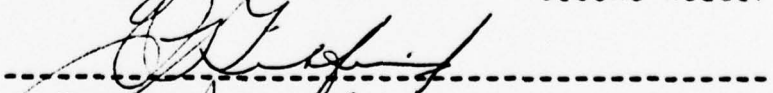
from the

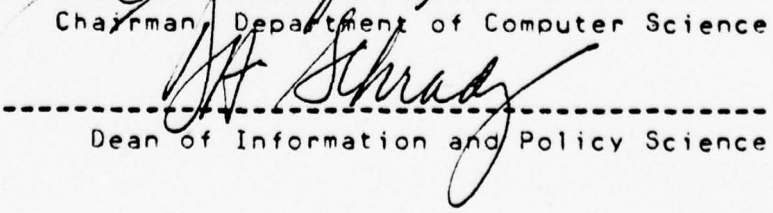
NAVAL POSTGRADUATE SCHOOL
DECEMBER 1976

Author 

Approved by: 
Thesis Advisor


Second Reader


Chairman, Department of Computer Science


Dean of Information and Policy Science

ABSTRACT

The design and implementation of an extension to the BASIC programming language for use on a microprocessor-based system has been described. The implementation is comprised of two subsystems, a compiler which generates code for a hypothetical zero-address machine and a run-time monitor which interprets this code. The design goals, solutions, and recommendations for further expansion of the system have been presented. The system was implemented in PL/M for use in a diskette-based environment.

CONTENTS

C. EXAMPLES OF BASIC-E PROGRAMS

NEXT J
RETURN

AKNOWLEDGEMENTS

Basic-E was an outgrowth of a class project for a compiler design course taught at the Naval Postgraduate School by Dr. Gary A. Kildall. The efforts of the members of this class provided the foundation for this undertaking. Subsequently, Glade Holyoak and Allan Craig assisted in the development of the project. Their contribution was substantial.

The continued support and assistance provided by Dr. Kildall was instrumental in the completion of this thesis. I extend my appreciation for his assistance and my respect for his knowledge of Computer Science.

Special thanks is also offered to the staff of the W. R. Church Computer Center, Naval Postgraduate School, for their patience and technical assistance.

Finally, the effort of Jane Foust in proof-reading this report is very much appreciated.

I. INTRODUCTION

A. HISTORY OF THE BASIC LANGUAGE

The Beginner's All-Purpose Symbolic Instruction Code (Basic) was developed at Dartmouth College to provide a simple, interactive language for casual computer users with applications in scientific computation [2]. To meet this goal, a limited vocabulary of instructions was included in the original definition of Basic. There was no concept of data typing and there were no default conditions to memorize. The interactive mode provided an ideal man/machine interface for creating and debugging programs, while the features of the language were well-suited for the expression of engineering problems. Since this type of environment satisfied the needs of a wide range of potential computer users, Basic was adapted by a number of universities and commercial firms. In particular, timesharing service bureaus saw the potential for expanded computer usage among non-computer specialists by providing its customers with the Basic language [10]. This led to the development of a number of dialects of Basic and to many extensions which satisfied specialized users.

As the use of Basic increased and extensions to the language proliferated, the need for standardization became a concern among computer specialists [12]. This concern led

to support multiple users by making the interpreter reenter-

to the formation, in 1974, of committee X3J2 of the American National Standards Committee which was tasked with developing a standard for the Basic programming language. The result of this effort was the Proposed American National Standards Institute (ANSI) standard for Minimal Basic [4]. This standard establishes a minimum set of features which should be included in the implementation of a Basic language processor. While the standard provides arithmetic and very simple string processing capabilities, it does not consider the more extensive features which led to the need for standardization in the first place. In a recent article by Lientz [9], the different commercially available Basic language processors were compared. This survey indicated that many Basic processors tend to provide similar features and include extensive facilities beyond those in the ANSI standard discussed above.

B. THE USE OF BASIC WITH MICROCOMPUTER SYSTEMS

Basic is becoming a widely used microprocessor application language. Typical of the many commercially available Basic interpreters is the Altair Basic [1]. Available in 4K, 8K, and 12K versions, it provides extensions which allow string and file processing and a wide range of predefined functions. The 12K version operates in conjunction with a floppy-disk system.

The IBM 5100 portable computer includes the Basic language implemented in read-only memory [5]. The language

and subroutine linkage. Other operations allow console and

provides stream data files, string manipulation including substring operations, matrix operators, and hard-copy output.

Although both of the Basic language processors described above include powerful extensions to the language, they have the following limitations. First, the entire source program must reside in memory at one time. This limits the size of programs which may be executed and thus discourages the use of remarks and indentation to show structure. Readability is limited by the restriction that identifiers consist of a single letter or a letter and a number. Finally, it is difficult for individuals to modify the system to support specific applications or devices.

C. OBJECTIVES OF THE BASIC-E LANGUAGE

Basic-E was designed to provide all the arithmetic processing features of the proposed standard for Basic as well as extensions and enhancements to the language. Extensions include multi-dimensional arrays, logical operators for numeric quantities, string manipulation, and sequential and random access to disk files. In addition, Basic-E retains the flavor of Dartmouth Basic while freeing the programmer from many of the original limitations. Such enhancements include improved control structures and features to increase readability. Basic-E also attempts to maintain compatibility with existing extensions to Basic where those extensions seem to have been accepted by the industry.

Similar to Altair Basic, Basic-E operates in conjunction with a disk operating system and requires at least 12K bytes of free memory. The CP/M monitor control program [3] was selected as the resident operating system because of its availability on a number of microcomputer systems, including those at the Naval Postgraduate School. CP/M is an interactive, single-user system providing standard I/O functions and supporting a named file system on IBM-compatible flexible disks. The system includes a text editor, dynamic debugger, symbolic assembler, and system utilities.

An additional goal of Basic-E was portability to other operating systems and backup storage devices other than the IBM-compatible format used with CP/M. To achieve this goal the programs were written with a separated I/O system in PL/M [6], a widely accepted system implementation language for 8080 microprocessors.

Basic-E provided a portable and expandable Basic language processing system incorporating the features discussed above. Unlike many existing implementations, Basic-E is not a purely interpretive language. A source program is compiled to pseudo machine code for the hypothetical Basic-E machine. This code is then executed interpretively by the run-time monitor. This approach is the same as used with Basic/M [8] an implementation of Basic with features similar to the proposed ANSI standard.

II. LANGUAGE SPECIFICATION

In the following section the Dartmouth Basic language will be reviewed, followed by a discussion of features of Basic-E which differ from the ANSI standard.

A. THE PROPOSED STANDARD FOR BASIC

1. Dartmouth Basic

Dartmouth Basic is a statement oriented language. Each statement consists of a line number and a command. Data are either real numeric or character string and no distinction is made between types of numeric data. An identifier terminated with a dollar sign refers to a string variable, and all other identifiers reference numeric quantities. Identifiers consist of one letter or a letter followed by a number. String variables consist of a single letter followed by a dollar sign. Arithmetic operations, performed on numeric data, are represented by the infix operators +, -, *, /, and ↑ (exponentiation) as well as the prefix operators + and -. Both data types may be compared using the infix relational operators <, <=, >, >=, and <> (not equal). One and two dimensional arrays are supported. The same identifier may refer to a subscripted and unsubscripted variable in the same program. Further, a dimension statement is not needed to specify a subscripted variable if

the value of a subscript does not exceed 10. Finally, a number of predefined functions perform elementary function evaluation.

2. The ANSI Standard

The proposed ANSI standard incorporates the features of Dartmouth Basic and also includes the following statements:

ON	RANDOMIZE	DEF
OPTION	STOP	

With the exception of the OPTION statement, most existing Basic implementations include all of the features described above. The OPTION statement is used to specify whether the lower bound of an array is zero or one.

Most existing Basic language processors go well beyond the ANSI standard to provide file-handling ability, formatted output, string manipulation, matrix operations, and a multitude of predefined functions. The survey by Lientz [9] documents these extensions for many large and mini-computer manufacturers, and for a number of timesharing services.

B. FEATURES OF THE BASIC-E LANGUAGE

Basic-E was designed to maintain compatibility with the proposed standard while extending the language to incorporate such features as string processing and disk file access. Enhancements were also included to provide

additional control structures and increased readability. In this section the features of Basic-E which do not appear in the ANSI standard will be discussed. Appendix I includes a complete description of the language.

1. Arithmetic Processing

Basic-E extended the arithmetic processing by supporting multiple dimensional arrays. However, all arrays must be dimensioned prior to use and the same identifier cannot refer to both a subscripted and unsubscripted variable.

Logical operators AND, OR, XOR (exclusive or), and NOT, were provided for numeric variables. The operations are performed on 32 bit two's complement binary representation of the integer portion of the variable.

User-defined functions may have any number of parameters including zero. The function must be defined prior to its use and, while it may refer to other functions, recursive references are not permitted.

2. Readability

Readability was improved by allowing variable names of any length, permitting free form input with statement continuation, and by not requiring all statements in the program to be labelled. Historically, Basic permitted variable names consisting of one letter or one letter and a number. This makes large programs difficult to understand

and debug. Basic-E allows variable names to be of any length but only the first 31 characters are considered unique. Basic traditionally has restricted a statement to one line. Basic-E provides a backslant (\) as a continuation character thus allowing many program lines to appear as one statement to the compiler.

3. Control Structure

The control structures included in standard Basic consist of the FOR, IF, GOTO, GOSUB and ON statements. Basic-E increased the power of the IF statement by providing an optional ELSE clause and by allowing a statement list to following the THEN and the ELSE. A statement list consists of executable statements separated by colons. Any executable statement may be included in the list with the exception of another IF.

4. String Processing

Basic-E contains features adequate for general string manipulation. Strings are created dynamically, vary in length to a maximum of 255 bytes, and may be subscripted. At any given time, a string occupies an amount of storage equal to its actual length plus one byte. The predefined LEN function returns the current length of a string. All string variables and string array elements are initialized null strings with a length of zero. Strings may be created and associated with a variable using the replacement operator (=), an INPUT statement, or a READ statement. Strings

entered from the console, appearing in a data statement, or read from a disk file may be either enclosed in quotation marks or delimited by a comma. Features of Basic-E allow concatenation of two strings to form a new string, comparison of string variables, and extraction of a string segment.

Concatenation of two string variables has been accomplished with the infix operator +. The new string length is the sum of the lengths of the strings being concatenated and must not exceed 255. Space is dynamically allocated for the new string as it is created.

Strings are compared with the same relational operators used for numeric data, using the ASCII collating sequence. Two strings are equal if and only if the strings have the same length and contain identical characters.

Substring extraction is accomplished using three predefined functions, LEFT$(A$,n), RIGHT$(A$,n), and MID$(A$,m,n). LEFT$ returns the string consisting of the first n characters of A$, while RIGHT$ returns the rightmost n characters of A$. MID$ is a general substring operator which returns the n characters of A$ beginning with character position m.

Other predefined functions are provided to facilitate processing strings. The CHR$ function converts a numeric quantity to a single ASCII character while STR$ converts a floating point number in internal form to a string representing its value.

User-defined functions may contain string parameters and, if the name of the function ends in a dollar sign, returns a string quantity.

5. Files

Data may be transferred between an Basic-E program and external devices using the file processing features of the language. The FILE statement identifies files and prepares them for access. The general form of a FILE statement is:

FILE <file name list>

where the file name is a string variable. If a file exists on the host file system with the name represented by the current value of the string variable then that file is opened. Otherwise, a file is created with that name. Each file is assigned a numeric identifier which is used for all further references to the file. An optional blocksize may be associated with the file. This identifies the file as a direct file with a specified record length. Data is transmitted between the file and the Basic-E machine using the READ and PRINT statements with the file option:

READ <file option> ; <read list>

PRINT <file option> ; <expression list>

The file option specifies the file desired by referencing the file identifier. An optional record identifier specifies the record desired when random access is used. Access to a file may be terminated by the CLOSE statement. Further, end of file processing is specified with the IF END

statement which has the following form:

IF END # <file identifier> THEN <label>

Files may be organized as either sequential or direct. Sequential files are a linear sequence of data items separated by commas or line terminators. Each reference to a sequential file retrieves the next data item or writes another data item. With each READ, the variables in the read list are assigned values from the file. Line terminators are treated as commas; there is no concept of a record as such. Likewise, with each PRINT, values from the expression list are writen to the file. The expressions are placed in the file as ASCII strings separated by commas except for the last data item in the list which is followed by a line terminator. The use of line terminators in this manner allow files to be displayed using system utilities and also allows files created with a text editor to be read by Basic-E programs.

A file declared with a specified blocksize is called a "direct file" and is made up of fixed length records. Each record consists of a collection of data items separated by commas. Individual records have line terminators as the last two bytes of the record. Note that direct files may be accessed sequentially or randomly. A READ statement with no read list will position the file to the selected record. In particular:

READ # 1, 1;

will rewind the file.

C. EXAMPLES OF BASIC-E PROGRAMS

Sample Basic-E programs are presented in this section which are intended to show features of the language described above.

1. Fibonacci Number Generator

```
REMARK  PROGRAM TO COMPUTE THE FIRST N
REMARK  FIBONACCI NUMBERS

    PRINT "THIS PROGRAM COMPUTES THE FIRST N"
    PRINT "FIBONACCI NUMBERS"
    PRINT "AN INPUT OF 0 TERMINATES THE PROGRAM"

FOR I = 1 TO 1 STEP 0  REMARK DO THIS FOREVER
    INPUT "ENTER THE VALUE OF N"; N
    IF N = 0 THEN \
        PRINT "PROGRAM TERMINATES" :\
        STOP
    IF N < 0 THEN \
        PRINT "N MUST BE POSITIVE. "; :\
        PRINT "PLEASE REENTER" \
    ELSE \
        GOSUB 300 REMARK CALCULATE AND PRINT RESULTS
NEXT I

300     REMARK SUBROUTINE TO CALCULATE FIB NUMBERS
    F1 = 1 : F2 = 1  REMARK INITIAL VALUES
    NUM = F1

REMARK HANDLE FIRST TWO NUMBERS (IF REQ) AS
REMARK SPECIAL CASES

    FOR J = 1 TO 2
        GOSUB 400
        IF N = 0 THEN \
            RETURN
    NEXT J

REMARK  HANDLE REMAINING NUMBERS

    FOR J = 1 TO 1 STEP 0
        NUM = F1 + F2
        GOSUB 400
        F2 = F1
        F1 = NUM
        IF N = 0 THEN \
            RETURN
```

```
    NEXT J
    RETURN

400     REMARK PRINT NEXT NUMBER AND
        REMARK DECREMENT N
    PRINT NUM, REMARK 5 TO A LINE
    N = N - 1
    RETURN

END
```

2. Label Generating Program

```
REMARK  PROGRAM BUILDS A FILE OF MAILING LABELS
REMARK  FROM A FILE CONTAINING 100 BYTE RECORDS
REMARK  WHICH CONTAIN NAME AND ADDRESS INFORMATION

    FILE    MASTER(100), LABELS
    IF END # 1 THEN 100

    FOR INDEX = 1 TO 1 STEP 0  REM UNTIL END OF FILE
        READ # 1; FIRST$, LAST$, STREET$, CITY$, STATE$,ZIP

        REMARK LINES ARE TRUNCATED AT 60 CHARACTERS

        LINE1$ = LEFT$(FIRST$ + " " + LAST$,60)
        LINE2$ = LEFT$(STREET$,60)

        REMARK INSURE ZIP NOT TRUNCATED

        LINE3$ = LEFT$(CITY$ + ", " + STATE$,54)
        LINE3$ = LINE3$ + " " + STR$(ZIP)

        PRINT # 2; LINE1$
        PRINT # 2; LINE2$
        PRINT # 2; LINE3$
    NEXT INDEX

100     PRINT "JOB COMPLETE"
        STOP
    END
```

III. IMPLEMENTATION

A. SYSIEM DESIGN

The Basic-E system is comprised of two subsystems, the compiler and the run-time monitor. The compiler includes a table-driven parser which checks statements for correct syntax and generates code for the Basic-E machine. This code is executed by the run time monitor. The simulated machine is a zero address stack computer which performs floating point arithmetic on 32 bit numbers, provides variable length string manipulation, and accesses sequential and random disk files.

The decision to compile the source program and then interpret the intermediate language was based on the following considerations. First, formal parsing techniques could be used to analyze the syntax of the source program making extensions to the language relatively easy. In this case a LALR parser-generator [14] was used to automatically generate the parse tables for the language. This makes extensions to the language relatively easy. Second, the entire source program does not reside in main memory during compilation. This provides the maximum amount of space for the symbol table and, perhaps more importantly, does not penalize the programmer for using comments and descriptive variable names. Finally, the run-time monitor can be modified

to support multiple users by making the interpreter reenterable at the end of each Basic-E machine cycle.

There are a number of considerations which dictate interpreting the intermediate language instead of compiling the source program to 8080 machine code. First, the majority of execution time is involved in evaluating floating point operations. Since this would be implemented as subroutine calls if the compiler generated machine code, the actual decrease in execution speed due to the interpreter is very small. Secondly, since the system, with the exception of the floating point package, is written in PL/M it is easily transportable to another microprocessor which supports PL/M. Extensive rewriting of the code generation would not be required.

The following sections discuss the design of the Basic-E machine and implementation of the compiler and run-time monitor. PL/M source listing of the programs are attached to this report.

B. THE BASIC-E MACHINE

1. Introduction

The Basic-E machine is a software simulation of a zero-address stack-processing computer, tailored to execute Basic programs. It is modeled after the ALGOL-E machine [11]. The Basic-E machine provides stack manipulation operations for arithmetic and string expression evaluation,

and subroutine linkage. Other operations allow console and file input/output, dynamic storage allocation for arrays and a variety of predefined functions. The Basic-E memory is divided into the several logical segments described below, including a free storage area, which is dynamically allocated by the run-time monitor. The size of the free storage area varies with the available space on the host system. A 16K system operating with CP/M provides five pages (256 byte blocks) of memory for the Basic-E machine.

2. Basic-E Machine Memory

The Basic-E machine memory is divided into a static section and a varying section. These sections are, in turn, divided into a number of logical segments as shown in Figure 1. The static section consists of memory locations which are not altered during program execution. The following segments make up the static section of memory:

a. The Floating-Point Data Area (FDA). The floating-point data area is used to store numeric constants defined within the source program. Values may be loaded directly onto the stack from the FDA.

b. The Code Area. The code segment consists of a sequence of Basic-E machine instructions, where each instruction is one byte in length. Certain instructions are followed by one or two bytes of data which may refer to the PRT described below. These instructions are referred to as literals and are distinguished

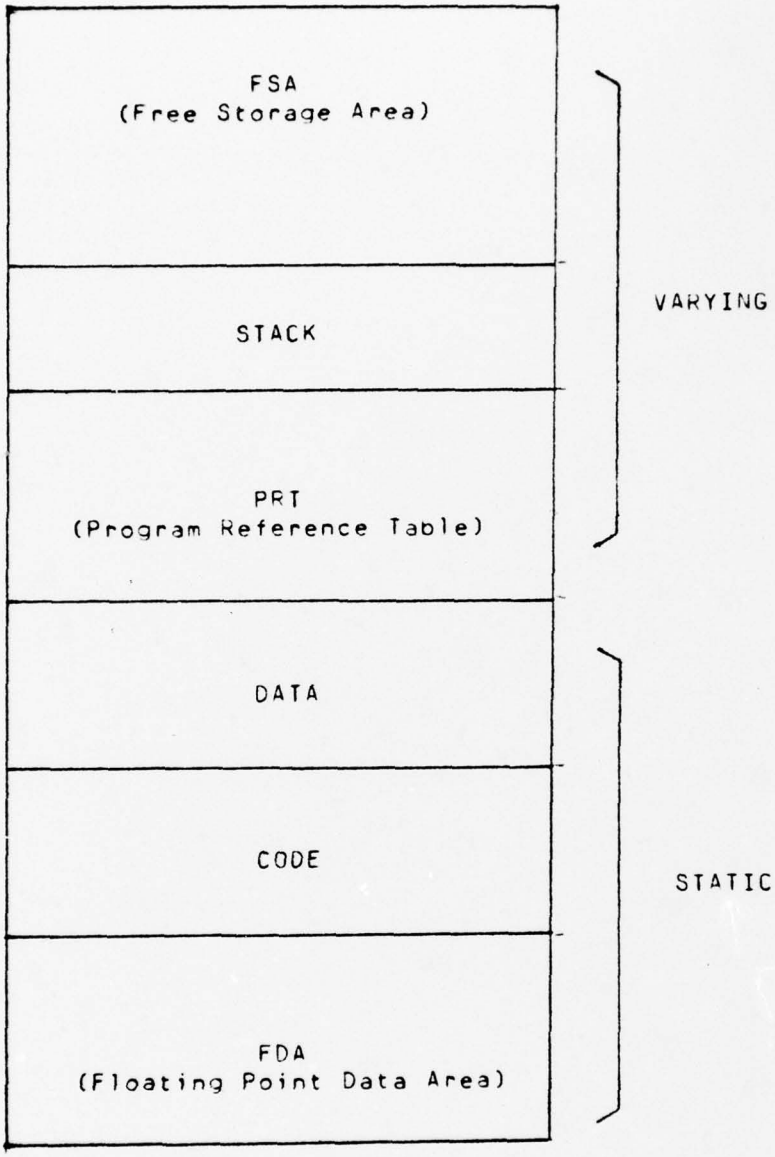


Figure 1

Structure of the NBASIC Machine

from regular instructions by having their leftmost bit set to one.

c. The Data Area. Floating-point and string constants defined in DATA statements are maintained in this section in the order in which they appear in the source program.

The varying section consists of memory locations which may be altered during program execution. The following segments make up this section of memory:

a. The Program Reference Table (PRT). The PRT stores the values of unsubscripted floating-point variables and pointers to subscripted floating-point variables and all string variables. Values may be loaded directly onto the stack from the PRT and into the PRT from the stack.

b. The Stack. The stack is used during program execution to evaluate expressions, hold return addresses for subroutine calls, and store values during input/output operations. Each stack element is four bytes wide. Numeric quantities are placed directly on the stack as 32 bit floating-point numbers. References to arrays are stored as address quatitities occupying the first two bytes of the element. Bytes three and four are not used in this case. Strings are also referenced by address. However, in the case of a string, byte three of the stack element is a flag used to indicate whether this string is a temporary string or currently assigned to a variable location in the PRT. This is necessary

to ensure that strings resulting from intermediate calculations are removed from the FSA. The stack is a circular queue which will hold 12 elements. Therefore it cannot owerflow but wraps around, overwriting itself.

c. The Free Storage Area (FSA). The FSA consists of the remaining memory space allotted by the host system. It is used to dynamically allocate arrays, string variables and file buffers. Figure 2 shows the organization of the free storage area.

3. The Basic-E Machine Architecture

The Basic-E machine consists of the memory space described above along with a set of registers whose functions are given below:

a. Register C. Register C, the program counter, contains the address of the next executable instruction.

b. Register D. Register D, the Data Area Pointer, is used to reference constants in the Data Area.

c. Register S. Register S contains the address of the current top of the stack.

c. Register A. Register A is a reference to the memory location addressed by register S.

d. Register B. Register B is a reference to the element on the stack below the element referenced by register S.

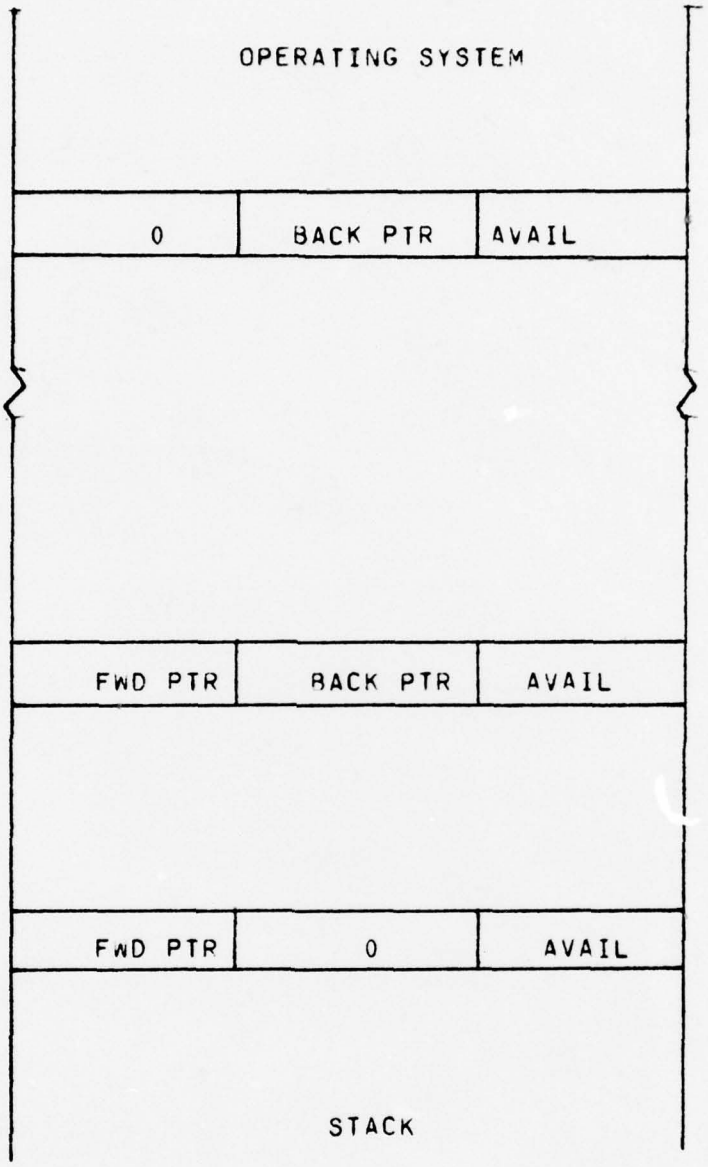


Figure 2

Organization of the Free Storage Area

a. Description of Machine Instructions

The Basic-E machine instructions are divided into the following categories: literal data references, arithmetic and boolean operators, program control operators, string operators, stack manipulation operators and built-in functions. All instructions consist of one eight bit operation code, possibly followed by one or two bytes of data. Most of the machine instructions will be described using the following notation:

a. The contents of a register are referenced as rX where X is register A, B, C or D.

b. The symbol := denotes assignment. For example rA := rA + 1 indicates that the contents of rA are incremented by one.

c. (rX) denotes the byte of data pointed to by register X. ((rX)) denotes the two bytes or address quantity pointed to by register X. PRT(rX) is a reference to the PRT cell referenced by the quantity in rX.

d. [X] is used to reference the string addressed by X.

e. POP is defined as rS:=rS-1. That is, the top element is eliminated from the stack. Likewise, PUSH is defined as rS:=rS+1.

f. PC1 is defined as rC:=rC+1. Likewise PCi is defined as rC:=rC+i.

b. Literal Data References

Literal data references are used to place integer constants on the stack. Literal string references place the address of a string constant on the stack. A literal data reference is any instruction in which the left-most bit is a one. All such literals occupy two bytes. If the second bit from the left is a zero (a LIT), the remaining 14 bits are placed on the stack as an address. Such quantities are generated by the compiler for control functions and may not be operated upon by the Basic-E machine arithmetic or logical operators. If the second bit from the left is a 1 (a LID), the remaining 14 bits refer to an entry in the PRT which is to be loaded onto the stack. A zero references the first element in the PRT, a one references the second element and so forth. A literal string reference is represented by the ILS machine operator followed by a one byte binary number which is the length of the string in bytes, followed directly by the string characters. The ILS operator is defined as follows:

ILS	inline string	PUSH. PC1; [rA]:=rC; rC:=rC+(rC)

c. Arithmetic Operators

The arithmetic and boolean operators are listed below along with their corresponding actions:

OPERATION	NAME	ACTION
FAD	floating add	rB:=rB+rA; POP. PC1

FMI	floating minus	rB:=rA-rB; POP. PC1
FMU	floating multiply	rB:=rA*rB; POP. PC1
FDI	floating divide	rB:=rA/rB. POP. PC1
EXP	exponentiation	rB:=rB↑rA. POP. PC1
NEG	floating negation	rA:= -rA. POP. PC1
LSS	floating less than	if rB<rA then rB:= -1
		else rB:= 0. POP. PC1
GTR	floating greater	if rB>rA then rB:= -1
	than	else rB:= 0. POP. PC1
EQU	floating equal	if rB=rA then rB:= -1
		else rB:= 0. POP. PC1
NEQ	floating unequal	if rB<>rA then rB:= -1
		else rB:= 0. POP. PC1
LEQ	floating less than	if rB<=rA then rB:= -1
	or equal	else rB:= 0. POP. PC1
GEQ	floating greater	if rB>=rA then rB:= -1
	than or equal	else rB:= 0. POP. PC1
NOT	logical not	rA:= NOT rA. PC1
OR	logical or	rB:=rB OR rA. POP. PC1
XOR	exclusive or	rB:=rB XOR rA. POP. PC1

d. String Operators

String operators allow comparsion and concatenation of variable length stings. Strings generated during program execution are placed in the free storage area, and strings are always referenced indirectly by placing the address on the stack. The string operators are listed below along with their corresponding actions:

OPERATION	NAME	ACTION
CAT	concatenate	[rB]:=[rB]+[rA]. POP. PC1
SEQ	string equal	if [rB]=[rA] then rB:= -1 else rB:= 0. POP. PC1
SNE	string not equal	if [rB]<>[rA] then rB:= -1 else rB:= 0. POP. PC1
SLT	string less than	if [rB]<[rA] then rB:= -1 else rB:= 0. POP. PC1
SLE	string less than or equal	if [rB]<=[rA] then rB:= -1 else rB:= 0. POP. PC1
SGT	string greater than	if [rB]>[rA] than rB:= -1 else rB:= 0. POP. PC1
SGE	string greater than or equal	if [rB]>=[rA] than rB:= -1 else rB:= 0. POP. PC1

e. Stack Operators

Stack operations bring elements to and from the stack, and allow manipulation of rA and rB. These operators are listed below with a description of their actions:

OPERATION	NAME	ACTION
CON	load constant	The two bytes following the operator are a reference to an element in the FDA which is to be placed on the stack. PC1
LOD	load variable	rA:=PRT(rA). PC1
STO	store nondestruct	PRT(rB):=rA. rB:=rA. POP.

		PC1
STD	store destructive	PRT(rB):=rA. POP. POP. PC1
STS	store string	if [PRT(rB)]<>null then release [PRT(rB)]. PRT(rB) :=rA. rB:=rA. POP. PC1
DEL	delete from stack	POP. PC1.
DUP	duplicate	PUSH. rA:=rB. PC1
XCH	exchange	<temp>:=rB. rB:=rA. rA:=<temp>; PC1

f. Program Control Operators

Program control operators provide for program termination, subroutine linkage and branching. The absolute branch (BRS) and conditional branch (BRC) instructions are followed by a two byte address which contains the branch address. In the case of the forward branches (BFN and BFC), the stack contains an increment to be added to the program counter. The program control operators are listed below along with their corresponding actions:

OPERATION	NAME	ACTION
XIT	terminate execution	
NOP	no operation	PC1
PRO	subroutine call	PUSH. rA:=rC+3; PC1; rC:=((rC))
RTN	return	rC:=rA; POP
BRS	unconditional branch	PC1; rC:=((rC))

BRC	conditional branch	if rA= 0 then PC1; rC:=((rC)) else rC:=rC+3; POP
BFN	branch forward	rC:=rC+rA. POP
BFC	conditional forward branch	if rB:= 0 then rC:=rC+rA else PC1. POP. POP

g. Input/Output Operators

The input/output operators provide data transfer between the console and the disk. Instructions are also provided to read constants from the data area. The definition of the operators is listed below:

OPERATION	NAME	ACTION
RCN	initiate console read	read console into buffer until end-of-line character found. PC1
RDV	read numeric from console	Push stack. Convert the next field in the console buffer to internal numeric and place it in rA. PC1
RES	read string from	Push stack. Place the next field from the console buffer into the FSA and put the put address in rA. PC1
ECR	end console read	Complete console read. Check for data remaining in the console buffer. PC1
WRV	write numeric to	Convert numeric in rA to a

	console	string and place it in the print buffer. POP. PC1
WST	write string to console	Place string referenced by rA in the print buffer. POP. PC1
DBF	dump print buffer	Write print buffer to console. PC1
NSP	space print buffer	Skip print buffer to next predefined tab. PC1
OPN	open disk file	Open disk file with name referenced by rA and block size in rB. Assign next file identifier to the file. POP. POP. PC1
CLS	close disk file	Close disk file whose file identifier is in rA. POP PC1
RRF	initiate random disk read	Ready to read disk file. rA contains record number, rB contains file identifier. POP. POP. PC1
RDB	initiate disk file for read	Ready to read sequentially from disk. rA contains file identifier. POP. PC1
RDN	read numeric from current disk file	PUSH. Place numeric field from selected disk file in rA. PC1
RDS	read string from	PUSH. Place string field from

	current disk file	selected disk in FSA and place address in rA. PC1
EDR	end disk read	Complete disk read. If the file is blocked, skip to the next line teminator. PC1
WRN	write numeric to disk	Convert numeric in rA to string and place in current disk record. POP. PC1
WRS	write string to disk	Place string addressed by rA in current disk record. POP. PC1
EDW	end disk write	Complete disk write. If the file is blocked, fill the remainder of the record with blanks and append a line terminator to the record. PC1
DEF	define end-of file	The two bytes following the operation code is a branch address where execution is to begin if end-of-file is detected on the file referenced by rA. POP. PC1.
DEF	define end of file	Two bytes following operation code is branch address if end of-file is detected on file referenced by rA. POP. PC3
RST	restore	The data area pointer is set to the beginning of the data

		area. PC1
DRS	read string from data area	PUSH. Place next data area field into the FSA and put address in rA. PC1
DRF	read numeric from data area	Push stack. Convert next field in data area to internal numeric and place in rA. PC1
OUT	output to port	The low order 8 bits of the value in rA is output to the 8080 machine port represented by rB. POP. POP. PC1
INP	input from port	Ra is set to the value input from the 8080 machine port represented by rA. PC1

h. Built-in Functions

Basic-E built-in functions perform complex operations which, in a real machine, might require a number of machine instructions. A description of these operations is given below.

OPERATION	NAME	ACTION
ROW	array setup	used to aid in setting-up an array in the FSA in row-major order. rA contains the number of dimensions. Below rA is each dimension. The lower bound is 0. PC1

SUB	subscript calculation	Used to compute the address of an array element in the FSA. The byte following the opcode is the number of dimensions. The indices are on the stack. PC1
CBA	convert to binary	The numeric value in rA is converted to a 16 bit binary value. PC1
ABS	absolute value	rA:=absolute value of rA PC1
INT	convert to integer	rA:=integer portion of rA PC1
RND	random number	PUSH. rA:=random number between 0 and 1. PC1
FRE	available space in FSA	PUSH. rA:=unused space in FSA. PC1
SGN	sign function	if rA>0 then rA:=1 else if rA< then rA:= -1 else rA:= 0. PC1
SIN	sine function	rA:=sine(rA); PC1
COS	cosine function	rA:=cosine(rA); PC1
ATN	arctangent function	rA:=arctan(rA); PC1
SQR	square root function	rA:= (rA). PC1
TAN	tangent function	rA:=tangent(rA); PC1
EXP	exponentiation	rA:=e raised to the rA power where e = 2.71828...

		PC1
COSH	hyperbolic cosine function	rA:=cosh(rA); PC1
SINH	hyperbolic sine function	rA:=sinh(rA); PC1
LOG	natural log function	rA:Ln(rA). PC1
POS	print position	PUSH. rA:= current position of the print buffer pointer. PC1
ASC	character convert	rA:= the ASCII numeric value of the first charachter of the string referenced by rA. PC1
CHR	string convert	The value in rA is converted to a string in the FSA. A reference to the string is placed in rA. PC1
TAB	tab print buffer	The print buffer pointer is set equal to rA mod 72. POP. PC1
LEFT	left substring	rA contains the number of bytes of the right portion of the string referenced by rB which will form a new string which is placed in the FSA. A reference to the string is placed in rB. POP.

		PC1
RIGHT	right substring	rA contains the number of bytes of the right portion of the string referenced by rB which will form a new string in the FSA. rB is set equal to the address of the new string. POP. PC1
MID	substring	Three parameters are on the stack. The first is the length of the substring to be created in the FSA. The second is the starting point into the string referenced by the third parameter. The top two elements are popped from the stack and rA is set equal to the address of the substring. PC1
RON	round to index	The floating point number in rA is rounded to the nearest integer and converted to a 16 bit address. PC1
CKO	check on index	rA contains the max number of labels in a ON statement. rB is the number of the selected label. If rB>rA or

		rB=0, then an error occurs otherwise POP. rA:=rA*3+1
BOL	beginning of line	rA contains the number of the line being executed. This value is saved for diagnostics. POP. PC1
ADJ	adjust branch address	rA:=rA+base of code area. PC1

C. COMPILER STRUCTURE

1. Compiler Organization

The compiler structure, diagrammed in Figure 3, requires two passes through the source program to produce an intermediate language file with optional source listing at the console. One pass writes all numeric constants to the INT file and determines the size of the code area, data area, and the PRT. These parameters are sent to the INT file at the end of pass one. By passing the numeric constants to the run-time monitor as unconverted ASCII strings, the compiler does not require the floating-point conversion package, saving considerable memory space. Pass two resolves forward references and outputs the generated code to the INT file.

2. Scanner

The scanner analyses the source program, returning a sequence of tokens to the parser. In addition, the scanner

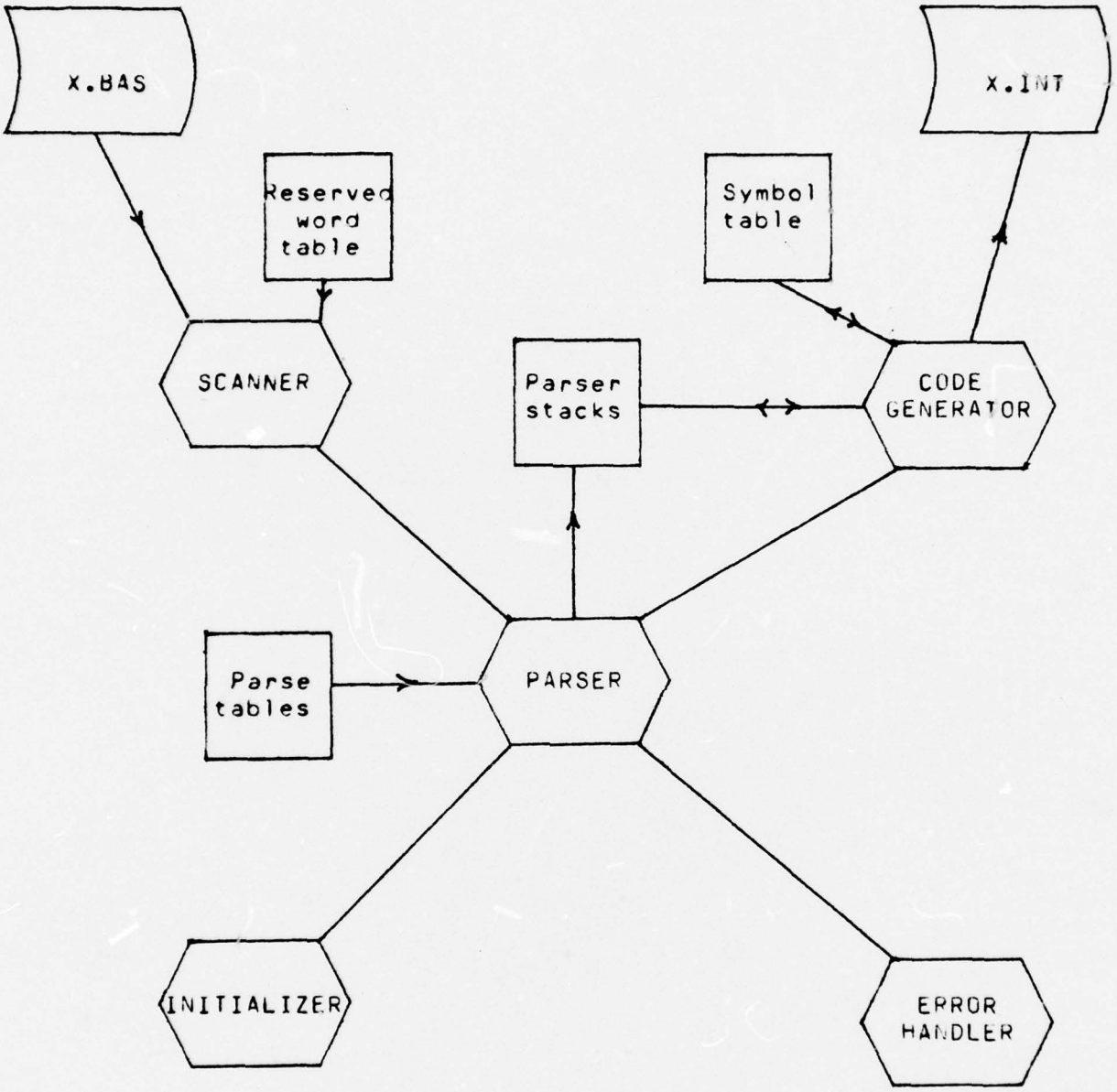


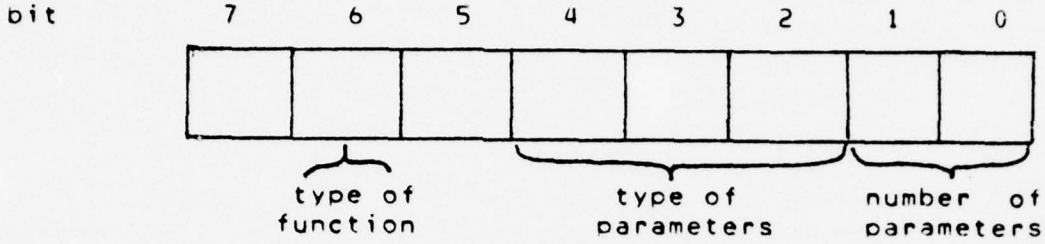
Figure 3

Basic-E Compiler Structure

provides the listing of the source file, skips remarks, processes data statements, sets and resets compiler toggles, and recognizes continuation characters. Analysis of the first non-blank character in the input stream determines the general class of the next token. The remainder of the token is then scanned, placing each successive character into the array ACCUM. The first byte of ACCUM contains the length of the token. The global variables TOKEN, SUBTYPE, FUNCOP, HASHCODE, and NEXTCHAR are set prior to returning to the parser. In the case of string constants whose length exceed the length of ACCUM, a continuation flag is set to allow the parser to obtain the remainder of the string.

If the scanner recognizes an identifier, it searches the reserved word table to determine if the identifier is a reserved word. If found, the token associated with that reserved word is returned to the parser. If the reserved word is a predefined function name, FUNCOP is set to the machine operation code for that function and SUBTYPE is set to provide additional information about the function, as shown in Figure 4.

Compiler toggles, statement continuation characters, listing of source lines, and data statements are handled by the scanner. Data statements processed by the scanner permits the string constants to appear as though read from the console, and thus they may or may not be delimited by quotation marks. In addition, constants defined in DATA statements can be located in a common area of the Basic-E machine



Type of parameter: 0 if numeric
1 if string

bit 2 is parameter 1
bit 3 is parameter 2
bit 4 is parameter 3

Type of function: 0 if numeric
1 if string

Figure 4

Subtype Field for Predefined Functions

which simplifies the run-time processing of READ statements. The penalty of not being able to syntax-check the constants is considered worth the gain in simplicity and flexibility.

3. Symbol Table

The symbol table stores attributes of program and compiler generated entities such as identifiers, function names, and labels. The information stored in the symbol table is built and referenced by the compiler to verify that the program is semantically correct and to assist in generating code. Access to the symbol table is provided through a number of subroutines operating on the symbol table global variables.

The symbol table is an unordered linear list of entries which grow toward the top of memory. Individual elements are accessed through a chained hash addressing scheme as diagrammed in Figure 5. Each entry in the hash table heads a linked list whose printnames all evaluate to the same hash address. A zero in the hash table indicates no entries exist on that particular chain. During references to the symbol table the global variable PRINTNAME contains the address of a vector containing the length of the printname followed by the printname itself. The global variable SYMHASH is set to the sum of the ASCII characters in the printname modulo 64. Entries are chained such that the most recent entry is the first element on the chain, but they physically appear in the symbol table sequentially in

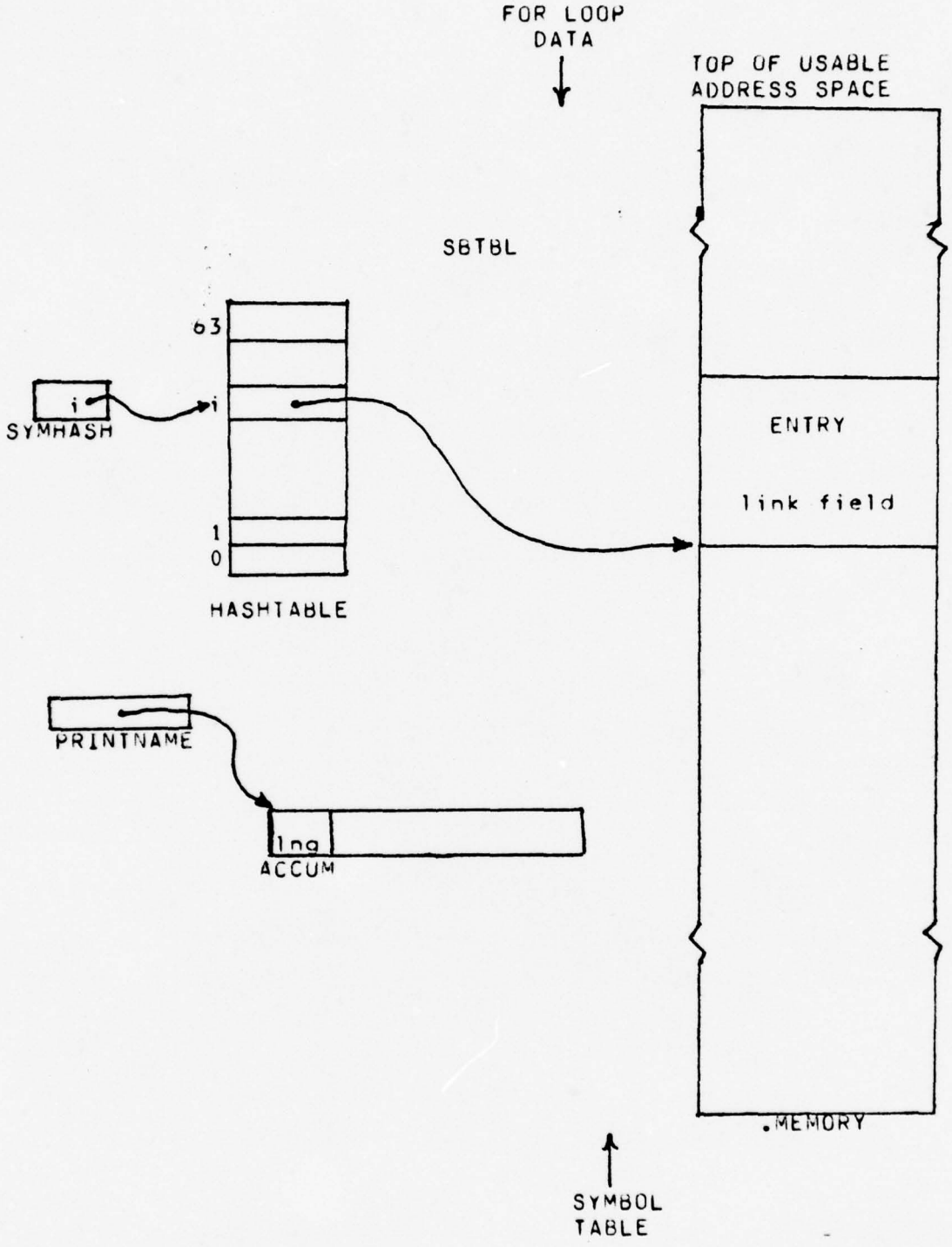


Figure 5

Symbol Table Organization

the order they are entered.

Each entry in the symbol table consists of a variable-length vector of eight entries. Figure 6 diagrams a typical entry. In the case of user-defined functions, the entry takes on a different format as shown in Figure 7. The parameters of a user-defined function are entered into the symbol table using the same format as a typical entry shown in Figure 6. To insure that the parameter names are local to the function definition, the entries for user-defined function parameters (if there are any parameters) are linked to the symbol table during code generation for the function and removed from the symbol table during the remainder of the pass. Since the paramaters appear directly after the entry for the function, a reference to a user-defined function accesses the parameters relative to the function name.

The symbol table is accessed using 11 primitive functions. LOOKUP is called with global variables SYMHASH and PRINTNAME set. If the printname is found, LOOKUP sets BASE to the beginning of the entry and returns true. Otherwise false is returned. ENTER also requires SYMHASH and PRINTNAME to be set and will build an entry placing it on the appropriate hash table chain. GETYPE, GETADDR, and GETSUBTYPE access fields in a particular symbol table entry while SETYPE, SETADDR, and SETSUBTYPE enter values in the corresponding fields. SETRES returns true if the address field has been resolved and false otherwise. RELINK and UNLINK provide local access to function parameters as

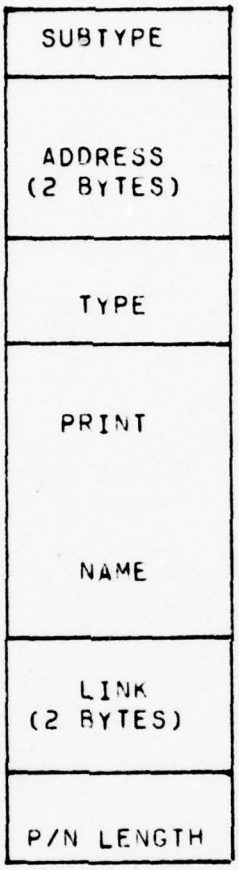


Figure 6

Typical Symboltable Entry

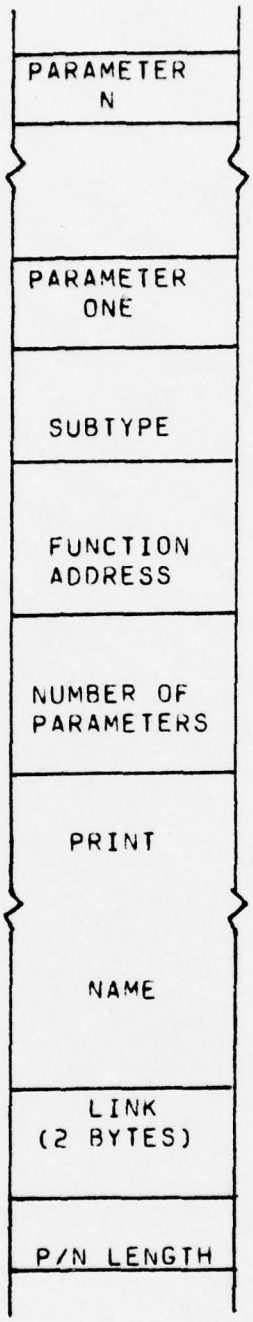


Figure 7

Symbol Table Entry For a User-Defined Function

discussed above. All the routines, with the exception of LOOKUP and ENTER, assume that BASE points to the proper entry. The symbol table vector is also used to maintain information required during FOR loop code generation. Each FOR loop uses eight bytes of the vector. This storage is allocated starting at the highest usable address in memory and builds toward the symbol table entries.

4. Parser

The parser is a table-driven pushdown automaton. It receives a stream of tokens from the scanner and analyzes them to determine if they form a sentence of the Basic-E grammar. As the parser accepts tokens, one of three actions will be performed. It may stack the token and continue to analyze the source program by fetching another token, or the parser may determine that it has recognized the right part of one of the productions of the language and cause a reduction to take place. Finally, the parser may determine that the current string of tokens does not produce a valid right part for a production and thus produces a syntax error message. The Basic-E grammar is designed so that each statement parses to a complete program causing a source program to appear as a series of programs. When an error is detected, the input characters are scanned until the end of the statement is found. The parser is then reinitialized, and the next "program" is parsed. The major data structures in the parser are the LALR parse tables and the parse stacks.

The parse stacks consist of a state stack and six auxiliary stacks. These auxiliary stacks are parallel to the parse stack and are used to store information required during code generation, such as token values, symbol table pointers, and temporary values associated with reductions.

5. Code Generation

In addition to verifying the syntax of source statements, the parser also acts as a transducer by associating semantic actions with reductions. Each time the parser determines that a reduction should take place, the procedure SYNTHESIZE is called with the number of the production passed as a parameter. The parse stacks contain the information required to perform the semantic action associated with the selected production. The action may include generation of Basic-E machine code and operations such as symbol table manipulations and updating of the parse stacks. Some productions have no semantic actions associated with them. In the following sections, the syntax of the language will be listed in the BNF notation followed by the semantic actions, offset with asterisks, associated with that production. The description will be in terms of compiler data structures and the Basic-E machine code generated. This notation is similar to that used in Ref. 11. For example production 57 would be described as follows:

```
<variable> ::= <subscript head> <expression> )
    *<subscript head>; <expression>;LID array name;
    *SUB
```

This indicates that code for the non-terminal <variable> is generated by first producing a <subscript head> and an <expression>, and then emitting a LID followed by the array name and a SUB. The reference to the array name would be stored in a parse stack.

a. Basic-E Language Structure

The overall structure of the Basic-E language is given by the following syntax equations:

```
(1) <program> ::= <line number> <statement> cr
        *<line number>; <statement>

(2) <line number> ::= <number>
        *<number>
(3)                 ¦<empty>

(4) <statemet> ::= <statement list>
        *<statement list>
(5)               ¦<if statement>
        *<if statement>
(6)               ¦<end statement>
        *<end statement>
(7)               ¦<dimension statement>
        *<dimension statement>
(8)               ¦<define statement>
        *<define statement>

(9) <statement list> ::= <simple statement>
        *<simple statement>
(10)                   ¦<statement list> : <simple statement>
        *<statement list>; <simple statement>

(11) <simple statement> ::= <let statement>
        *<let statement>
(12)                        ¦<assignment>
        *<assignment>
(13)                        ¦<for statement>
        *<for statement>
(14)                        ¦<next statement>
        *<next statement>
(15)                        ¦<file statement>
        *<file statement>
(16)                        ¦<close statement>
        *<close statement>
(17)                        ¦<read statement>
        *<read statement>
```

```
(18)                        ¦<print statement>
        *<print statement>
(19)                        ¦<goto statement>
        *<goto statement>
(20)                        ¦<gosub statement>
        *<gosub statement>
(21)                        ¦<input statement>
        *<input statement>
(22)                        ¦<stop statement>
        *<stop statement>
(23)                        ¦<return statement>
        *<return statement>
(24)                        ¦<on statement>
        *<on statement>
(25)                        ¦<restore statement>
        *<restore statement>
(26)                        ¦<randomize statement>
        *<randomize statement>
(27)                        ¦<out statement>
        *<out statement>
(28)                        ¦<empty>
```

b. Assignment Statements and Expressions

The following productions generate code for assignment statements and expressions. The types of operands which are legal for each of the binary operators is shown in the Table 1. The operand for the unary operators +, -, and NOT must be numeric quantities. A check is made to insure the above semantic rules are followed.

Checks are also made to insure that subscripted variables are dimensioned before being used, that they have the correct number of subscripts, that each subscript is of type numeric, and that a subscripted variable is not used as a FOR loop index. Likewise, checks are made on the number and type of parameters in a function call to insure they match the function definition.

```
(29) <let statement> ::= <assignment>
        *<assignment>
```

```
(39) <assignment> ::= <assign head> <expression>
        *<assign head>; <expression>; if type of
        *expression string then STS otherwise STD

(31) <assign head> ::= <variable> =
        *<variable>; if simple variable then
        *LIT <variable>

(32) <expression> ::= <logical factor>
        *<logical factor>
(33)                  |<expression> <or> <expression>
        *<expression>; <expression>; <or>

(34) <xor> ::= OR
        *BOR
(35)         |XOR
        *XOR

(36) <logical factor> ::= <logical secondary>
        *<logical secondary>
                      |<logical factor> AND <logical factor>
        *<logical factor>; <logical factor>; AND

(38) <logical secondary> ::= <logical primary>
        *<logical primary>
(39)                        |NOT <logical secondary>
        *<logical secondary>; NOT

(40) <logical primary> ::= <arithmetic expression>
        *<arithmetic expression>
(41)                       |<arithmetic expression> <relation>
                             <arithmetic expression>
        *<arithmetic expression>; <arithmetic expression>;
        *<relation>

(42) <arithmetic expression> ::= <term>
        *<term>
                               |<arithmetic expression> + <term>
        *<arithmetic expression>; <term>;
        *if string then CAT else FAD
(44)                          |<arithmetic expression> - <term>
        *<arithmetic expression>; <term>; FMI
(45)                            |+ <term>
        *<term>
(46)                            |- <term>
        *<term> NEG

(47) <term> ::= <primary>
        *<primary>
(48)          |<term> * <primary>
        *<term>; <primary>; FMU
(49)          |<term> / <primary>
        *<term>; <primary> FDI

(50) <primary> ::= <element>
```

```
        *<element>
(51)                ¦<primary> ↑ <element>
        *<primary>; <element>; EXP

(52) <element> ::= <variable>
        *<variable>;
        *if simple variable then LID <variable>
        *otherwise LOD
(53)                ¦<constant>
        *<constant>
(54)                ¦<function call>
        *<function call>
(55)                ¦( <expression> )
        *<expression>

(56) <variable> ::= <identifier>
        *<identifier>; {place <identifier> in the
        *symbol table if first reference and set
        *the type to simple}
(57)                 ¦<subscript head> <expression> )
        *<subscript head>; <expression>;
        *LID array name; SUB

(58) <subscript head> ::= <identifier> (
        *<identifier>; {check that <identifier> has
        *been previously dimensioned and save
        *for future use}
(59)                      ¦<subscript head> <expression> ,
        *<subscript head>; <expression>

(60) <function call> ::= <function head> <expression> )
        *<function head>; <expression>;
        *for user-defined function if type <expression>
        string then STS otherwise STD; PRO
        *for predefined function function name
        *where function name was saved in production 65
(61)                      ¦<function name>
        *for user-defined function PRO otherwise
        *function name where function name was saved in
        *production 65

(62) <function head> ::= <function name> (
        *<function name>;
        *if user-defined function then LIT parameter address
        *where parameter address is determined from
        *symbol table entry for the function.
(63)                     ¦<function name> <expression> ,
        *<function name>; <expression>; if user-defined then
        *if type <expression> string then STS otherwise STD
        *LIT <parameter address>
        *where <parameter address> is determined from
        *symbol table entry for the function.

(64) <function name> ::= <user-defined name>
        *{check that <user defined name> is in the
```

```
        *symbol table}
(65)                    ¦<predefined name>
        *{save predefined name for future use}

(66) <constant> ::= <number>
        *<number>; CON next constant location
        *if pass 1 spool to INT file
(67)                ¦<string>
        *ILS <string>

(68) <relation> ::= =
        *EQU
(69)                ¦GE
        *GEQ
(70)                ¦>=
        *GEQ
(71)                ¦<=
        *LEQ
(72)                ¦LE
        *LEQ
(73)                ¦>
        *GTR
(74)                ¦<
        *LSS
(75)                ¦<>
        *NEQ
(76)                ¦NE
        *NEQ
```

c. Control Statements

The control statements in the Basic-E language are given by the following syntax equations:

```
(77) <for statement> ::= <for head> TO <expression>
                              <step clause>
        *<for head>; <expression>; <step clause>;
        *if stepclause then DUP;
        *LID <index>;  FAD;
        *if stepclause then LIT <index>; XCH;
        *STO;
        *if stepclause XCH; LIT 0; LSS; LIT 5; BFC; LEQ;
        *if stepclause LIT 2; BFN;
        *FEQ; BRC EXIT; CODE:

(78) <for head> ::= <for> <assignment>
        *<for>; <assignment>; BRS CODE; TEST:
        *save the <identifier> for use later}

(79) <for> ::= FOR
        *{set forstatement flag true}

(80) <step clause> ::= STEP <expression>
```

```
        *<expression>; {set stepclause true}
(81)        ¦<empty>
        *{set stepclause false};
        *LIT <index>; CON 0

(143) <next statement> ::= <next head> <identifier>
        *<next head>; <identifier>; BRS TEST;
        *EXIT:
(144)                      ¦NEXT
        *BRS TEST; EXIT:

(145) <next head> ::= NEXT
(146)                 ¦<next head> <identifier> ,
        *BRS TEST; EXIT:

(82) <if statement> ::= <if group>
        *<if group>; END:
(83)                  ¦<if else group> <statement list>
        *<if else group>; <statement list>; END:
(84)                  ¦IF END # <expression> THEN <number>
        *<expression>; {resolve label}; DEF; LABEL:

(85) <if group> ::= <if head> <statment list>
        *<if head>; <statement list>; END:
(86)              ¦<if head> <number>
        *<if head>; {resolve label}; BRS LABEL

(87) <if else group> ::= <if head> <statement list> ELSE
        *<if head>; <statement list>; ELSE:; BRS END

(88) <if head> IF <expression> THEN
        *<expression>; BRC END

(131) <goto statement> ::= <goto> <number>
        *<goto>; {resolve label}; BRS LABEL

(132) <on statement> ::= <on goto> <label list>
        *<on goto>; <label list>; {save number of labels
        *in <label list> for later use}
(133)                    ¦<on gosub> <label list>
        *<on gosub>; <label list>; {save number of labels
        *in <label list> for later use}; END:

(134) <on goto> ::= <expression> <goto>
        *<expression>; <goto>; RON; LIT number of labels;
        *CKO; BFN

(135) <ongosub> ::= <expression> <gosub>
        *<expression>; <gosub>; LIT END; ADJ; XCH; RON;
        *LIT number of labels; CKO; BFN

(136) <label list> ::= <number>
        *{resolve label}; BRS LABEL

(137)                ¦<label list> , <number>
```

```
        *<label list>; {resolve label}; BRS LABEL

(138) <gosub statement> ::= <gosub> <number>
        *<gosub>; {resolve label}; PRO LABEL

(139) <goto> ::= GOTO
(140)          |GO TO

(141) <gosub> ::= GOSUB
(142)          |GO SUB

(148) <return statement> ::= RETURN
        *RTN

(149) <stop statement> ::= STOP
        *XIT
```

d. Declaration Statements

The declaration statements in the Basic-E language are given by the following syntax equations:

```
(89) <define statement> ::= <ud function name>
                              <dummy arg list> = <expression>
        *<ud function name>; <dummy arg list>;
        *<expression>; XCH; RTN; END:;
        *{unlink dummy arguments from symbol table}

(90) <ud function name> ::= DEF <userdefined name>
        *BRS END; {if pass2 relink dummy arguments into
        *symbol table}

(91) <dummy arg list> ::= <dummy arg head> <identifier>
        *<dummy arg head>; <identifier>; {enter dummy
        *argument in symbol table}
(92)                    |<empty>

(93) <dummy arg head> ::= (
(94)                    |<dummy arg head> <identifier> ,
        *<dummy arg head>; <identifier>; {enter dummy
        *argument in symbol table}

(95) <file statement> ::= <file head> <file declaration>
        *<file head>; <file declaration>

(96) <file head> ::= FILE
(97)               |<file head> <file declaration> ,
        *<file head> <file declaration>

(98) <file declaration> ::= <identifier> <file rec size>
        * <file rec size>; {if <identifier> is not in
        *the symbol table enter it}; LID <identifier>; OPN
```

```
(99) <file rec size> ::= ( <expression> )
        *<expression>
                        ¦<empty>
        *LIT 0

(101) <dimension statement> ::= DIM <dimension variable list>
        *<dimension variable list>

(102) <dimension variable list> ::= <dimension variable>
        *<dimension variable>; ROW subscript count; STD
(103)                           ¦<dimension variable list> ,
                                    <dimension variable>
        *<dimension variable list>; <dimension variable>;
        *ROW subscript count; STD

(104) <dimension variable> ::= <dim var head> <expression> )
        *<dim var head>; <expression>; {increment count of
        *subscripts and save for future use}; RON

(105) <dim var head> ::= <identifier> (
        *{enter <identifier> in symbol table};
        *LIT <identifier>
(106)                   ¦<dim var head> <expression> ,
        *<dim var head>; <expression>; {increment count of
        *subscripts and save for future use}; RON
```

e. Input/Output Statements

The input/output statements in the Basic-E language are given by the following syntax equations:

```
(107) <close statement> ::= CLOSE <close list>
        *<close list>

(108) <close list> ::= <expression>
        *<expression>; CLS
(109)                ¦<close list> , <expression>
        *<close list>; <expression>; CLS

(110) <read statement> ::= READ <file option> <readlist>
        *<file option>; <read list>; EDR

(111)                    ¦READ <read list>
        *<read list>

(112) <input statement> ::= INPUT <prompt option> <read list>
        *<prompt option>; <read list>; ECR; {set inputstmt
        *inputstmt false}

(113) <prompt option> ::= <constant> ;
        *<constant>; WST; DBF; RCN; {set inputstmt true}

(114)                   ¦<empty>
```

```
          *DBF; RCN; {set inputstmt true}

(115) <read list> ::= <variable>
          *<variable> ; code from table 2

(116)                 ¦<read list> , <variable>
          *<read list> ; <variable>; code from table 2
(117)                 ¦<empty>

(118) <print statement> ::= PRINT <print list> <print end>
          *<print list>; <print end>

(119)                       ¦PRINT <file option> <file list>
          *<file option>; <file list>; EDW; {set fileio false}

(120) <print list> ::= <expression>
          *<expression>; if string WST otherwise WRV
(121)               ¦<print list>; ¢print delim>; <expression>
          *<print list>; <print delim>; <expression>;
          *if string WST otherwise WRV
(122)                 ¦<empty>

(123) <file list> ::= <expression>
          *<expression>; if string WRS otherwise WRN
(124)                 ¦<file list> , <expression>
          *<expression>; <expression>; if string WRS
          *otherwise WRN

(125) <print end> ::= <print delim>
          *<print delim>
(126)                 ¦<empty>
          *DBF

(127) <file option> ::= # <expression> ;
          *<expression>; RON; RDB; {set fileio true}
(128)                  ¦# <expression> , <expression>
          *<expression>; <expression>; RDF; RON; XCH; RON;
          *{set fileio true}

(129) <print delim> ::= ;
(130)                  ¦,
          *NSP

(147) <out statement> ::= OUT <expression> , <expression>
          *<expression>; <expression>; RON; XCH; RON; OUT
```

Table 1

Permissible Variable Types With Binary Operators

	string	numeric
string	type 1, +	error
numeric	error	type 1, type 2, +

type 1 operands		type 2 operands	
<	>=	-	or
<=	<>	*	and
>	=	/	xor
= (assignment)		↑	

Table 2

Code Generation For Input/Output

	string	numeric
input statement	RES	RDV
file read or print	RDS	RDN
data area read	DRS	DRF

D. RUN-TIME MONITOR STRUCTURE

1. Organization

The Run-Time Monitor consists of three modules, the initializer, the interpreter, and the floating point package. The initial organization of memory is shown in Figure 11a. Execution of a Basic-E program is accomplished by passing the name of an INT file to the BUILD initializer program. The Basic-E machine is then constructed above the BUILD program in memory, and control is passed to the interpreter. The entire Basic-E machine is repositioned to reside above the interpreter and execution of the Basic-E machine code begins. Execution continues until a XIT instruction is encountered, or a control-z is entered in response to an input statement.

2. Initializer Program

The initializer program sets up the floating point package and then opens the INT file of the program to be executed. The Basic-E machine is constructed by creating the FDA, Code Area and Data area. The numeric constants appear first on the INT file separated by asterisks. Each constant is converted to the internal floating point representation and stored in the FDA. The list of constants is terminated by a dollar sign.

Three 16 bit binary numbers follow the constants in the INT file, which represent the size of the code area,

size of the data area, and the number of entries in the PRT. This allows the BUILD program to determine the absolute address of each Section of the Basic-E machine. The addresses of the machine sections are passed to the interpreter through fixed locations in the floating point package. If sufficient memory is available, the generated code is read from the INT file and placed in either the Data section or the Code section of the machine. Constants from DATA statements are placed in the data area. All other instruction are put in the code area. In the case of BRS, BRC, PRO, CON, and DEF instructions, the address following the instructions is relocated to reflect actual machine addresses. The BUILD program continues reading the INT file until a 7FH instruction code is encountered. Control is then passed to the interpreter.

3. Interpreter

The interpreter repositions the Basic-E machine so that the space occupied by the BUILD program may be reused (Figure 11b). The FSA and Basic-E machine registers are initialized and then the problem program is executed by interpreting the Basic-E machine instructions. The major data structures in the interpreter are the FSA, stack, console buffer, and print buffer.

The organization of the FSA was outlined in section 3. Three primitive functions are provided to manipulate the linked lists. GETSPACE(N) returns the address of a block of

N consecutive bytes of storage using a first-fit algorithm. RELEASE places a block of storage back into the pool of available storage. AVAILABLE returns the number of bytes of storage available in the FSA.

Arrays, file buffers, and strings (with the exception of string constants defined in the program) are dynamically allocated in the FSA. Each allocated block of storage has an AVAIL byte associated with it (see Figure 2). When the storage contains a string the AVAIL byte indicates the number of variables which are referencing the string at a particular time. For example, execution of the following program segment:

```
X$ = "A STRING"
FOR I = 1 TO 100
    A$(I) = X$
NEXT I
```

would allocate storage for X$ but then each assignment of X$ to an element of A$ would increment the AVAIL counter and not create a new copy of the string. If the AVAIL byte reaches 255 a new copy of the string is created. When an assignment to a string variable takes place and the string previously associated with that variable is in the FSA (it also may be a constant or null) the AVAIL byte of the old string is decremented and if it is 1 the space occupied by the string is released.

The Basic-E machine stack is implemented as a four byte wide circular stack. The top two elements are rA and

r8. All access to the stack is in terms of these pointers. Primitive operations are provided to push and pop the stack, interchange the top two elements, and load a value onto the stack.

The input buffer is a temporary storage for characters entered from the console. The entire line is read by CP/M and placed into the buffer. Individual values are then extracted as required for RDV and RDS machine instructions.

The print line buffer is used to store characters as an output line is developed. After all data for a print line has been placed in the buffer, the data line is printed. A buffer-pointer is used to keep track of the next available position where a character can be placed. The buffer-pointer may be repositioned with the NSP and TAB instructions; the buffer is emptied by executing a DBF instruction or when the buffer-pointer exceeds the end of the print buffer.

4. Floating-Point Package

The floating point package consists of a set of subroutines written in 8080 assembly language which perform arithmetic, function evaluation, and conversion operations on 32 bit floating point numbers. The package was obtained from the Intel User Library [8].

IV. RECOMMENDATIONS FOR EXTENSIONS TO BASIC-E

There are a number of potential extensions to the Basic-E language which could be made. They include formatted input/output, a TRACE statement for debugging, and additional string processing features.

Basic processors have traditionally implemented formatted input/output by modifying the print statement as shown below:

PRINT USING <format string> ; <expression>

The format string contains a description of the format into which the values in the expression list are to be placed. A disadvantage of including formatted I/O in Basic-E is the amount of memory required to interpret the format strings.

A TRACE instruction would list the source program line numbers as each statement was executed and optionally print the current value of selected variables. An accompanying UNTRACE statement would disable the trace.

Additional string operators could include a search function which would determine the position of one string within another, and a substring replacement operation which would replace a substring with another (possibly null) string.

The above additions to the Basic-E language were not incorporated because of the limited time available to complete the project.

A desirable modification to the design of the Basic-E Run-Time Monitor would be segmenting the interpreter into modules and then loading only those modules which where actually required for the execution of the program. This would provide more memory for the Basic-E machine without loss of capability. Possible segments might include a base segment with the numeric processing functions, a transcendental functions segment, a string processing segment, and a file handling segment. Special purpose features, such as matrix operations and plotting routines, could easily be included as segments.

APPENDIX I - BASIC-E LANGUAGE MANUAL

Elements of BASIC-E are listed in alphabetical order in this section of the manual. A synopsis of the element is shown, followed by a description and examples of its use. The intent is to provide a reference to the features of this implementation of BASIC and not to teach the Basic Language.

A program consists of one or more properly formed BASIC-E statements. An END statement, if present, terminates the program, and additional statements are ignored. The entire ASCII character set is accepted, but all statements may be written using the common 64-character subset.

In this section the "synopsis" presents the general form of the element. Square brackets [] denote an optional feature while braces {} indicate that the enclosed section may be repeated zero or more times. Terms enclosed in < > are either non-terminal elements of the language, which are further defined in this section, or terminal symbols. All special characters and capitalized words are terminal symbols.

ELEMENT:

ABS predefined function

SYNOPSIS:

ABS (<expression>)

DESCRIPTION:

The ABS function returns the absolute value of the <expression>. The argument must evaluate to a floating point number.

EXAMPLES:

```
ABS(X)

ABS(X*Y)
```

ELEMENT:

ASC predefined function

SYNOPSIS:

ASC (<expression>)

DESCRIPTION:

The ASC function returns the ASCII numeric value of the first character of the <expression>. The argument must evaluate to a string. If the length of the string is 0 (null string) an error will occur.

EXAMPLES:

```
ASC(A$)

ASC("X")

ASC(RIGHT$(A$,7))
```

ELEMENT:

ATN predefined function

SYNOPSIS:

ATN (<expression>)

DESCRIPTION:

The ATN function returns the arctangent of the <expression>. The argument must evaluate to a floating point number.

EXAMPLES:

```
ATN(X)

ATN(SQR(SIN(X)))
```

PROGRAMMING NOTE:

All other inverse trigonometric functions may be computed from the arctangent using simple identities.

ELEMENT:

CHR$ predefined function

SYNOPSIS:

CHR$ (<expression>)

DESCRIPTION:

The CHR$ function returns a character string of length 1 consisting of the character whose ASCII equivalent is the <expression> converted to an integer modulo 128. The argument must evaluate to a floating point number.

EXAMPLES:

```
CHR$(A)

CHR$(12)

CHR$((A+B/C)*SIN(X))
```

PROGRAMMING NOTE:

CHR$ can be used to send control characters such as a linefeed to the output device. The following statement would accomplish this:

```
PRINT CHR$(10)
```

ELEMENT:

CLOSE statement

SYNOPSIS:

[<line number>] CLOSE <expression> {, <expression>}

DESCRIPTION:

The CLOSE statement causes the file specified by each <expression> to be closed. Before the file may be referenced again it must be reopened using a FILE statement.
An error occurs if the specified file has not previously appeared in a FILE statement.

EXAMPLES:

```
    CLOSE 1

150 CLOSE I, K, L*M-N
```

PROGRAMMING NOTE:

On normal completion of a program all open files are closed. If the program terminates abnormally it is possible that files created by the program will be lost.

ELEMENT:

<constant>

SYNOPSIS:

[<sign>] <integer> . [<integer>] [E <sign> <exp>]

["] <character string> ["]

DESCRIPTION:

A <constant> may be either a numeric constant or a string constant. All numeric constants are stored as floating point numbers. Strings may contain any ASCII character.

Numeric constants may be either a signed or unsigned integer, decimal number, or expressed in scientific notation. Numbers up to 31 characters in length are accepted but the floating point representation of the number maintains approximatly seven significant digits (1 part in 16,000,000). The largest magnitude that can be represented is approximately 3.6 times ten to the 38th power. The smallest non-zero magnitude that can be represented is approximately 2.7 times ten to the minus 39th power.

String constants may be up to 255 characters in length. Strings entered from the console, in a data statement, or read from a disk file may be either enclosed in quotation marks or delimited by a comma. Strings used as constants in the program must be enclosed in quotation marks.

EXAMPLES:

10

-100.75639E-19

"THIS IS THE ANSWER"

ELEMENT:

COS predefined function

SYNOPSIS:

COS(<expression>)

DESCRIPTION:

COS is a function which returns the cosine of the <expression>. The argument must evaluate to a floating point number expressed in radians.
A floating point overflow occurs if the absolute value of the <expression> is greater than two raised to the 24th power times pi radians.

EXAMPLES:

```
COS(B)

COS(SQR(X-Y))
```

ELEMENT:

COSH predefined function

SYNOPSIS:

COSH (<expression>)

DESCRIPTION:

COSH is a function which returns the hyperbolic cosine of the <expression>. The argument must evaluate to a floating point number.

EXAMPLES:

```
COSH(X)

COSH(X↑2+Y↑2)
```

ELEMENT:

DATA statement

SYNOPSIS:

[<line number>] DATA <constant> {, <constant>}

DESCRIPTION:

DATA statements define string and floating point constants which are assigned to variables using a READ statement. Any number of DATA statements may occur in a program. The constants are stored consecutively in a data area as they appear in the program and are not syntax checked by the compiler.
Strings may be enclosed in quotation marks or optionally delimited by commas.

EXAMPLES:

```
10  DATA 10.0,11.72,100

    DATA "XYZ",11.,THIS IS A STRING
```

ELEMENT:

DEF statement

SYNOPSIS:

[<line number>] DEF <function name> (<dummy argument list>) = <expression>

DESCRIPTION:

The DEF statement specifies a user defined function which returns a value of the same type as the <function name>. One or more <expressions> are passed to the function and used in evaluating the expression. The passed values may be in string or floating point form but must match the type of the corresponding dummy argument. Recursive calls are not permitted. The <expression> in the define statement may reference <variables> other than the dummy arguments, in which case the current value of the <variable> is used in evaluating the <expresssion>. The type of the function must match the type of the <expression>.

EXAMPLES:

```
10  DEF FNA(X,Y) = X + Y - A

    DEF FNB$(A$,B$) = A$ + B$ + C$

    DEF FN.COMPUTE(A,B) = A + B - FNA(A,B) + D
```

ELEMENT:

DIM statement

SYNOPSIS:

[line number] DIM <identifier> (<subscript list>)
{,<identifier> (<subscript list>)}

DESCRIPTION:

The dimension statement dynamically allocates space for floating point or string arrays. String array elements may be of any length up to 255 bytes and change in length dynamically as they assume different values. Initially, all floating point arrays are set to zero and all string arrays are null strings. An array must be dimensioned explicitly; no default options are provided. Arrays are stored in row major order.
Expressions in subscript lists are evaluated as floating point numbers and rounded to the nearest integer when determining the size of the array. All subscripts have an implied lower bound of 0.
When array elements are referenced a check is made to ensure the element resides in the referenced array.

EXAMPLES:

```
DIM A(10,20), B(10)

DIM B$(2,5,10),C(I + 7.3,N),D(I)

DIM X(A(I),M,N)
```

PROGRAMMING NOTE:

A <DIM statement> is an executable statement, and each execution will allocate a new array.

ELEMENT:

END statement

SYNOPSIS:

[line number] END

DESCRIPTION:

An END statement indicates the end of the source program. It is optional and, if present, it terminates reading of the source program. If any statments follow the END statement they are ignored.

EXAMPLES:

```
10  END

    END
```

ELEMENT:

EXP (<expression>)

DESCRIPTION:

The EXP function returns e (2.71828....) raised to the power of the <expression>. The argument must evaluate to a floating point number.
If the value of the <expression> exceeds two to the 127th power, a floating point overflow occurs.

EXAMPLES:

```
EXP(X)

EXP(LOG(X))
```

ELEMENT:

<expression>

DESCRIPTION:

Expressions consist of algebraic combinations of variables, constants, and operators. The hierarchy of operators is:

1) ()
2) ↑
3) *, /
4) +, -, concat (+), unary +, unary -
5) relational ops <, <=, >, >=, =, <>
   LT, LE, GT, GE, EQ, NE
6) NOT
7) AND
8) OR, XOR

Relational operators result in a 0 if false and -1 if true. NOT, AND, and OR are performed on 32 bit two's complement binary representation of the integer portion of the variable. The result is then converted to a floating point number. String variables may be operated on by relational operators and concatenation only. Mixed string and numeric operations are not permitted.

EXAMPLES:

```
X + Y

A$ + B$

(A <= B) OR (C$ > D$) / (A - B AND D)
```

ELEMENT:

FILE statement

SYNOPSIS:

```
[<line number>] FILE <variable> [(<expression>)]
                {, <variable>[(<expression>)]}
```

DESCRIPTION:

A file statement opens files used by the program. The order of the names determines the numbers used to reference the files in READ and PRINT statements. The value assigned to the first simple variable is file1, the second is file 2, and so forth. There may be any number of FILE statements in a program, but there is a limit to the number of files which may be opened at one time. Currently this limit is set at 6 files. The optional <expression> designates the logical record length of the file. If no length is specified, the file is written as a continuous string of fields with carriage return linefeed characters separating each record. If the record length is present, a carriage return linefeed will be appended to each record. The <variable> must not be subscripted and it must be of type string.

EXAMPLES:

```
FILE INPUT$, OUTPUT$

FILE TABLE.INC$, TAX.INC$(160), PAY.AMT.DAY$(N*3-J)
```

PROGRAMMING NOTE:

The run-time monitor will always assign the lowest available (not previously assigned) number to the file being opened. Thus if files are closed and others opened it is possible that number assignment may vary with program flow.

ELEMENT:

FOR statement

SYNOPSIS:

[<line number>] FOR <index> = <expression> TO
<expression> [STEP <expression>]

DESCRIPTION:

Execution of all statements between the FOR statement and its corresponding NEXT statement is repeated until the indexing variable, which is incremented by the STEP <expression> after each iteration, reaches the exit criteria. If the step is positive, the loop exit criteria is that the index exceeds the value of the TO <expression>. If the step is negative, the index must be less than the TO <expression> for the exit criteria to be met.
The <index> must be an unsubscripted variable and is initially set to the value of the first <expression>. Both the TO and STEP expressions are evaluated on each loop, and all variables associated with the FOR statement may change within the loop. If the STEP clause is omitted, a default value of 1 is assumed. A FOR loop is always executed at least once. A step of 0 may be used to loop indefinitely.

EXAMPLES:

```
FOR I = 1 TO 10 STEP 3

FOR INDEX = J*K-L TO 10*SIN(X)

FOR I = 1 TO 2 STEP 0
```

PROGRAMMING NOTE:

If a step of 1 is desired the step clause should be omitted. The execution will be substantially faster since less runtime checks must be made.

ELEMENT:

FRE predefined function

SYNOPSIS:

FRE

DESCRIPTION:

The FRE function returns the number of bytes of unused space in the free storage area.

EXAMPLE:

```
FRE
```

FUNCTION:

<function name>

SYNOPSIS:

FN<identifier>

DESCRIPTION:

Any <identifier> starting with FN refers to a user-defined function. The <function name> must appear in a DEF statement prior to being used in an <expression>. There may not be any spaces between the FN and the <identifier>.

EXAMPLES:

```
FNA

FN.BIGGER.$
```

ELEMENT:

GOSUB statement

SYNOPSIS:

[<line number>] GOSUB <line number>
[<line number>] GO SUB <line number>

DESCRIPTION:

The address of the next sequential instruction is saved on the run-time stack, and control is transferred to the subroutine labeled with the <line number> following the GOSUB or GO SUB.

EXAMPLES:

```
10  GOSUB   300

    GO SUB  100
```

PROGRAMMING NOTE:

The max depth of GOSUB calls allowed is controlled by the size of the run-time stack which is currently set at 12.

ELEMENT:

GOTO statement

SYNOPSIS:

[<line number>] GOTO <line number>

[<line number>] GO TO <line number>

DESCRIPTION:

Execution continues at the statement labeled with the <line number> following the GOTO or GO TO.

EXAMPLES:

```
100 GOTO 50

    GO TO 10
```

ELEMENT:

<identifier>

SYNOPSIS:

<letter> { <letter> or <number> or . } [$]

DESCRIPTION:

An identifier begins with an alphabetic character followed by any number of alphanumeric characters, or periods. Only the first 31 characters are considered unique. If the last character is a dollar sign the associated variable is of type string, otherwise it is of type floating point.

EXAMPLES:

A

B$

XYZ.ABC

PAY.RECORD.FILE.NUMBER.76

PROGRAMMING NOTE:

All lowercase letters appearing in an <identifier> are converted to uppercase unless compiler toggle D is set to off.

ELEMENT:

IF statement

SYNOPSIS:

[<line number>] IF <expression> THEN <line number>

[<line number>] IF <expression> THEN <statement list>

[<line number>] IF <expression> THEN <statement list>
ELSE <statement list>

DESCRIPTION:

If the value of the <expression> is not 0 the statements which make up the <statement list> are executed. Otherwise the <statement list> following the ELSE is executed, if present, or the next sequential statement is executed.
In the first form of the statement if the <expression> is not equal to 0, an unconditional branch to the label occurs.

EXAMPLES:

```
IF A$ < B$ THEN X= Y*Z

IF (A$<B$) AND (C OR D) THEN 300

IF B THEN X = 3.0 : GOTO 200

IF J AND K THEN GOTO 11 ELSE GOTO 12
```

ELEMENT:

IF END statement

SYNOPSIS:

[<line number>] IF END #<expression> THEN <line number>

DESCRIPTION:

If during a read to the file specified by the <expression>, an end of file is detected control is transferred to the statement labeled with the line number following the THEN.

EXAMPLES:

```
        IF END # 1 THEN 100

    10  IF END # FILE.NUMBER - INDEX THEN 700
```

PROGRAMMING NOTE:

On transfer to the line number following the THEN the stack is restored to the state prior to the execution of the READ statement which caused the end of file condition.

ELEMENT:

INP predefined function

SYNOPSIS:

INP (<expression>)

DESCRIPTION:

The INP function performs an input operation on the 8080 machine port represented by the value of the <expression> modulo 256 returning the resulting value. The argument must evaluate to a floating point number.

EXAMPLES:

```
INP(2)

INP(CURRENT.INPUT.PORT)
```

ELEMENT:

INPUT statement

SYNOPSIS:

```
[<line number>] INPUT [<prompt string> ;]
                <variable> {, <variable> }
```

DESCRIPTION:

The <prompt string>, if present, is printed on the console. A line of input data is read from the console and assigned to the variables as they appear in the variable list. The data items are separated by commas and/or blanks and terminated by a carriage return. Strings may be enclosed in quotation marks. If a string is not enclosed by quotes, the first comma terminates the string. If more data is requested than was entered, or if insufficient data items is entered, a warning is printed on the console and the entire line must be reentered.

EXAMPLES:

```
10  INPUT A,B

    INPUT "SIZE OF ARRAY?"; N

    INPUT "VALUES?"; A(I),B(I),C(A(I))
```

PROGRAMMING NOTE:

Trailing blanks in the <prompt string> are ignored. One blank is always supplied by the system.

ELEMENT:

INT predefined function

SYNOPSIS:

INT (<expression>)

DESCRIPTION:

The INT function returns the largest integer less than or equal to the value of the <expression>. The argument must evaluate to a floating point number.

EXAMPLES:

```
INT (AMOUNT / 100)

INT(3 * X * SIN(Y))
```

ELEMENT:

LEFT$ predefined function

SYNOPSIS:

LEFT$ (<expression> , <expression>)

DESCRIPTION:

The LEFT$ function returns the n leftmost characters of the first <expression>, where n is equal to the integer portion of the second <expression>. An error occurs if n is negative. If n is greater than the length of the first <expression> then the entire expression is returned. The first argument must evaluate to a string and the second to a floating point number.

EXAMPLES:

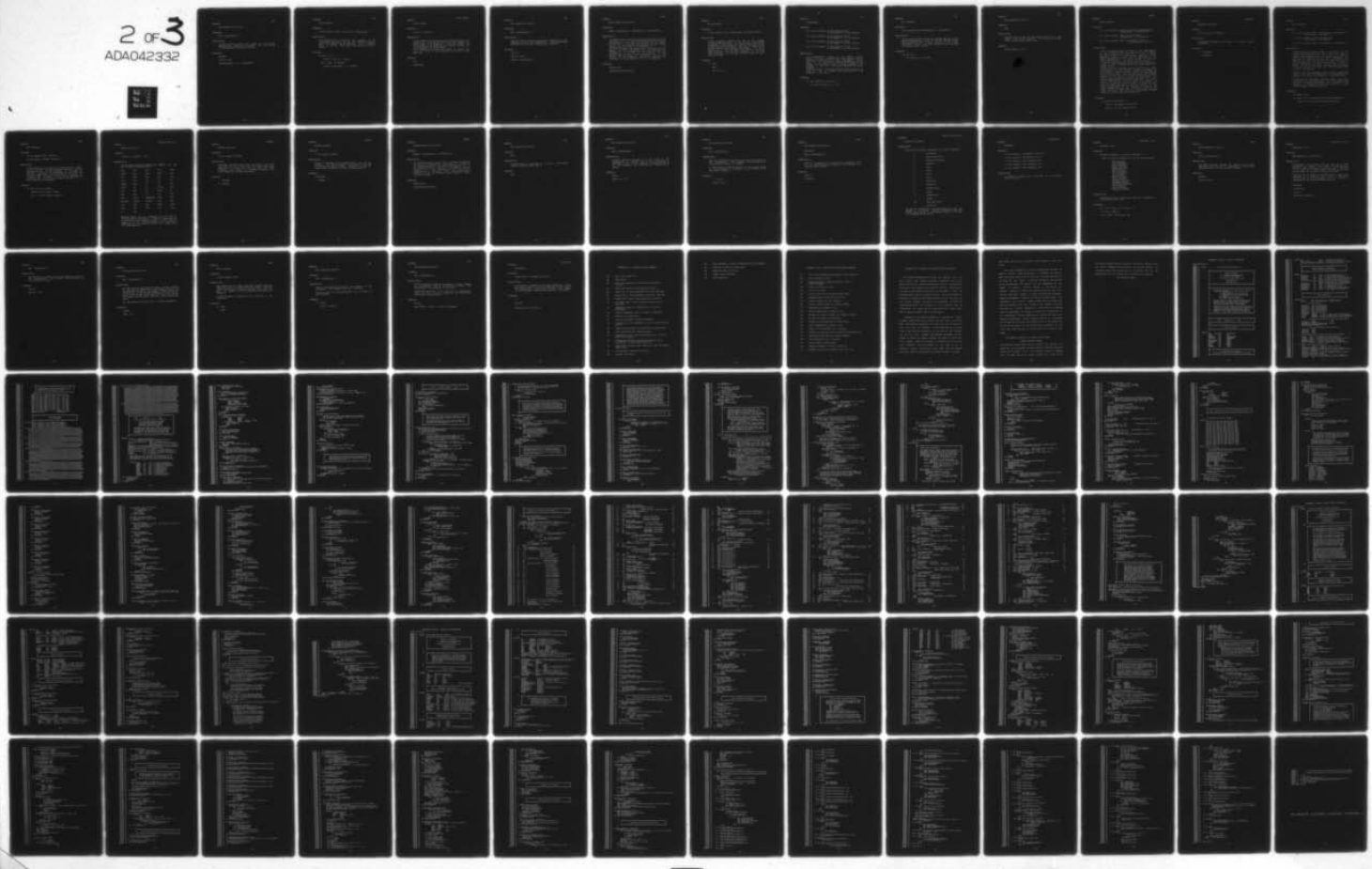
```
LEFT$ (A$,3)

LEFT$(C$+D$,I-J)
```

AD-A042 332 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
A MICROPROCESSOR IMPLEMENTATION OF EXTENDED BASIC.(U)
DEC 76 G E EUBANKS

UNCLASSIFIED NL

2 OF 3
ADA042332



ELEMENT:

LEN predefined function

SYNOPSIS:

LEN (<expression>)

DESCRIPTION:

The LEN function returns the length of the string <expression> passed as an argument. Zero is returned if the argument is the null string.

EXAMPLES:

```
LEN(A$)

LEN(C$ + B$)

LEN(LASTNAME$ + "," + FIRSTNAME$)
```

ELEMENT:

LET statement

SYNOPSIS:

[<line number>] [LET] <variable> = <expression>

DESCRIPTION:

The <expression> is evaluated and assigned to the <variable> appearing on the left side of the equal sign. The type of the <expression>, either floating point or string, must match the type of the <variable>.

EXAMPLES:

```
100 LET A = B + C

    X(3,A) = 7.32 * Y + X(2,3)

 73 W = (A<B)  OR (C$>D$)

    AMOUNT$ = DOLLARS$ + "." + CENTS$
```

<line number>

ELEMENT:

<line number>

SYNOPSIS:

<digit> { <digit> }

DESCRIPTION:

<line numbers> are optional on all statements and are ignored by the compiler except when they appear in a GOTO, GOSUB, or ON statement. In these cases, the <line number> must appear as the label of one and only one <statement> in the program.
<line numbers> may contain any number of digits but only the first 31 are considered significant by the compiler.

EXAMPLES:

100

4635276353

ELEMENT:

LOG predefined function

SYNOPSIS:

LOG (<expression>)

DESCRIPTION:

The LOG function returns the natural logarithm of the absolute value of the <expression>. The argument must evaluate to a non-zero floating point number.

EXAMPLES:

```
LOG (X)

LOG((A + B)/D)

LOG10 = LOG(X)/LOG(10)
```

ELEMENT:

MID$ predefined function

SYSNOPSIS:

MID$ (<expression> , <expression> , <expression>)

DESCRIPTION:

The MID$ function returns a string consisting of the n characters of the first <expression> starting at the mth character. The value of m is equal to the integer portion of the second <expression> while n is the integer portion of the third <expression>.
The first argument must evaluate to a string, and the second and third arguments must be floating point numbers. If m is greater than the length of the first <expression> a null string is returned. If n is greater than the number of characters left in the string all the characters from the mth character are returned. An error occurs if m or n is negative.

EXAMPLES:

```
MID$(A$,I,J)

MID$(B$+C$,START,LENGTH)
```

ELEMENT:

NEXT statement

SYNOPSIS:

[<line number>] NEXT [<identifier> {,<identifier>}]

DESCRIPTION:

A NEXT statement denotes the end of the closest *unmatched* FOR statement. If the optional <identifier> is present it must match the index variable of the FOR statement being terminated. The list of <identifiers> allows matching multiple FOR statements. The <line number> of a NEXT statement may appear in an ON or GOTO statement, in which case execution of the FOR loop continues with the loop variables assuming their current values.

EXAMPLES:

```
10  NEXT

    NEXT I

    NEXT I, J, K
```

ELEMENT:

ON statement

SYNOPSIS:

(1) [<line number>] ON <expression> GOTO
<line number> {, <line number>}

(2) [<line number>] ON <expression> GO TO
<line number> {, <line number>}

(3) [<line number>] ON <expression> GOSUB
<line number> {, <line number>}

(4) [<line number>] ON <expression> GO SUB
<line number> {, <line number>}

DESCRIPTION:

The <expression>, rounded to the nearest integer value, is used to select the <line number> at which execution will continue. If the <expression> evaluates to 1 the first <line number> is selected and so forth. In the case of an ON ... GOSUB statement the address of the next instruction becomes the return address.
An error occurs if the <expression> after rounding is less than one or greater than the number of <line numbers> in the list.

EXAMPLES:

```
10  ON I GOTO 10, 20, 30, 40

    ON J*K-M GO SUB 10, 1, 1, 10
```

ELEMENT:

OUT statement

SYNOPSIS:

[<line number>] OUT <expression> , <expression>

DESCRIPTION:

The low-order eight bits of the integer portion of the second <expression> is sent to the 8080 machine output port selected by the integer portion of the first expression modulo 256. Both arguments must evaluate to floating point numbers.

EXAMPLES:

```
100 OUT 3,10

    OUT PORT.NUM, NEXT.CHAR
```

ELEMENT:

POS predefined function

SYNOPSIS:

POS

DESCRIPTION:

The POS function returns the current position of the output line buffer pointer. This value will range from 1 to the print buffer size.

EXAMPLE:

```
PRINT TAB(POS + 3);X
```

ELEMENT:

PRINT statement

SYNOPSIS:

(1) [<line number>] PRINT #<expression>,<expression>;
<expression> {, <expression> }

(2) [<line number>] PRINT # <expression> ;
<expression> {, <expression> }

(3) [<line number>] PRINT <expression> <delim>
{ <expression> <delim> }

DESCRIPTION:

A PRINT statement sends the value of the expressions in the expression list to either a disk file (type(1) and (2)) or the console (type (3)). A type (1) PRINT statement sends a random record specified by the second <expression> to the disk file specified by the first <expression>. An error occurs if there is insufficient space in the record for all values.
A type (2) PRINT statement outputs the next sequential record to the file specified by the <expression> following the #.
A type (3) PRINT statement outputs the value of each <expression> to the console. A space is appended to all numeric values and if the numeric item exceeds the right margin then the print buffer is dumped before the item is printed. The <delim> between the <expressions> may be either a comma or a semicolon. The comma causes automatic spacing to the next tab position (14,28,42,56). If the current print position is greater than 56 then the print buffer is printed and the print position is set to zero. A semicolon indicates no spacing between the printed values. If the last <expression> is not followed by a <delim> the print buffer is dumped and the print position set equal to zero. The buffer is automatically printed anytime the print position exceeds 71.

EXAMPLES:

```
100 PRINT #1;A,B,A$+"*"

    PRINT # FILE,WHERE; A/B,D,"END"

    PRINT A, B, "THE ANSWER IS"; x
```

ELEMENT:

RANDOMIZE statement

SYNOPSIS:

[<line number>] RANDOMIZE

DESCRIPTION:

A RANDOMIZE statement initializes the random number generator.

EXAMPLES:

```
10  RANDOMIZE

    RANDOMIZE
```

ELEMENT:

READ statement

SYNOPSIS:

(1) [<line number>] READ <expression>, <expression>;
<variable> {, <variable> }

(2) [<line number>] READ # <expression>;
<variable> {, <variable> }

(3) [<line number>] READ # <variable> {, <variable> }

DESCRIPTION:

A READ statement assigns values to variables in the variable list from either a file (type (2) and (3)) or from a DATA statement (type (1)). Type (2) reads a random record specified by the second expression from the disk file specified by the first expression and assigns the fields in the record to the variables in the variable list. Fields may be floating point or string constants and are delimited by a blank or comma. Strings may optionally be enclosed in quotes. An error occurs if there are more variables than fields in the record.

The type (3) READ statement reads the next sequential record from the file specified by the expression and assigns the fields to variables as described above.

A type (2) READ statement assigns values from DATA statements to the variables in the list. DATA statements are processed sequentially as they appear in the program. An attempt to read past the end of the last data statement produces an error.

EXAMPLES:

```
100 READ A,B,C$

200 READ # 1,I; PAY.REG,PAY.OT,HOURS.REG,HOURS.OT

    READ # FILE.NO; NAME$,ADDRESS$,PHONE$,ZIP
```

ELEMENT:

REM statement

SYNOPSIS:

[<line number>] REM [<remark>]

[<line number>] REMARK [<remark>]

DESCRIPTION:

A REM statement is ignored by the compiler and compilation continues with the statement following the next carriage return. The REM statement may be used to document a program. REM statements do not affect the size of program that may be compiled or executed. An unlabeled REM statement may follow any statement on the same line. And the <line number> may occur in a GOTO, GOSUB or ON statement.

EXAMPLES:

```
10  REM THIS IS A REMARK

    REMARK THIS IS ALSO A REMARK

    LET X = 0 REM INITIAL VALUE OF X
```

ELEMENT:

reserved word list

SYNOPSIS:

<letter> { <letter> } [$]

DESCRIPTION:

The following words are reserved by BASIC-E and may not be used as <identifiers>:

ABS	AND	ASC	ATN	CHR$
CLOSE	COS	COSH	DATA	DEF
DIM	ELSE	END	EQ	EXP
FILE	FOR	FRE	GE	GO
GOSUB	GOTO	GT	IF	INP
INPUT	INT	LE	LEFT$	LEN
LET	LOG	LT	MID$	NE
NEXT	NOT	ON	OR	OUT
POS	PRINT	RANDOMIZE	READ	REM
RESTORE	RETURN	RIGHT$	RND	SGN
SIN	SINH	SQR	STEP	STOP
STR$	SUB	TAB	TAN	THEN
TO	VAL			

Reserved words must be preceeded and followed by either a special character or a space. Spaces may not be embedded within reserved words. Unless compiler toggle D is set, lowercase letters are converted to uppercase prior to checking to see if an <identifier> is a reserved word.

ELEMENT:

RESTORE statement

SYNOPSIS:

[<line number>] RESTORE

DESCRIPTION:

A RESTORE statement repositions the pointer into the data area so that the next value read with a READ statement will be the first item in the first DATA statement. The effect of a RESTORE statement is to allow rereading the DATA statements.

EXAMPLES:

```
    RESTORE

10  RESTORE
```

ELEMENT:

RETURN statement

SYNOPSIS:

[<line number>] RETURN

DESCRIPTION:

Control is returned from a subroutine to the calling routine. The return address is maintained on the top of the run-time monitor stack. No check is made to insure that the RETURN follows a GOSUB statement.

EXAMPLES:

```
130 RETURN

    RETURN
```

ELEMENT:

RIGHT$ predefined function

SYNOPSIS:

RIGHT$ (<expression> , <expression>)

DESCRIPTION:

The RIGHT$ function returns the n rightmost characters of the first <expression>. The value of n is equal to the integer portion of the second <expression>. If n is negative an error occurs; if n is greater than the length of the first <expression> then the entire <expression> is returned. The first argument must produce a string and the second must produce a floating point number.

EXAMPLES:

```
RIGHT$(X$,1)

RIGHT$(NAME$,LNG.LAST)
```

ELEMENT:

RND predefined function

SYNOPSIS:

RND

DESCRIPTION:

The RND function generates a uniformly distributed random number between 0 and 1.

EXAMPLE:

```
RND
```

ELEMENT:

SGN predefined function

SYNOPSIS:

SGN (<expression>)

DESCRIPTION:

The SGN function returns 1 if the value of the <expression> is greater than 0, -1 if the value is less than 0 and 0 if the value of the <expression> is 0. The argument must evaluate to a floating point number.

EXAMPLES:

```
SGN(X)

SGN(A - B + C)
```

ELEMENT:

SIN predefined function

SYNOPSIS:

SIN (<expression>)

DESCRIPTION:

SIN is a predefined function which returns the sine of the <expression>. The argument must evaluate to a floating point number in radians.

A floating point overflow occurs if the absolute value of the <expression> is greater than two raised to the 24th power times pi.

EXAMPLES:

```
X = SIN(Y)

SIN(A - B/C)
```

ELEMENT:

SINH predefined function

SYNOPSIS:

SINH (<expression>)

DESCRIPTION:

SINH is a function which returns the hyperbolic sine of the <expression>. The argument must evaluate to a floating point number.

EXAMPLES:

SINH(Y)

SINH(B<C)

ELEMENT:

special characters

DESCRIPTION:

The following special characters are used by BASIC-E:

↑	circumflex
(	open parenthesis
)	closed parenthesis
*	asterisk
+	plus
-	minus
/	slant
:	colon
;	semicolon
<	less-than
>	greater-than
=	equal
#	number-sign
,	comma
CR	carriage return
\	backslant

Any special character in the ASCII character set may appear in a string. Special characters other than those listed above, if they appear outside a string, will generate an IC error.

ELEMENT:

<statment>

SYNOPSIS:

[<line number>] <statement list> <cr>

[<line number>] IF statement <cr>

[<line number>] DIM statement <cr>

[<line number>] DEF statement <cr>

[<line number>] END statement <cr>

DESCRIPTION:

All BASIC-E statments are terminated by a carriage return (<cr>).

<statement list>

ELEMENT:

<statment list>

SYNOPSIS:

<simple statement> {: <simple statement> }

where a <simple statement> is one of the following:

FOR statement
NEXT statement
FILE statement
CLOSE statement
GOSUB statement
GOTO statment
INPUT statement
LET statement
ON statement
PRINT statement
READ statement
RESTORE statement
RETURN statement
RANDOMIZE statement
OUT statement
STOP statement
<empty> statement

DESCRIPTION:

A <statement list> allows more than one <statement> to occur on a single line.

EXAMPLES:

```
LET I = 0 : LET J = 0 : LET K = 0

X = Y+Z/W : RETURN

:::::: PRINT "THIS IS OK TOO"
```

ELEMENT:

STR$ predefined function

SYNOPSIS:

STR$ (<expression>)

DESCRIPTION:

The STR$ function returns the ASCII string which represents the value of the <expression>. The argument must evaluate to a floating point number.

EXAMPLES:

```
STR$(X)

STR$(3.141617)
```

<subscript list>

ELEMENT:

<subscript list>

SYNOPSIS:

<expression> {, <expression> }

DESCRIPTION:

A <subscript list> may be used as part of a <DIM statement> to specify the number of dimensions and extent of each dimension of the array being declared or as part of a <subscripted variable> to indicate which element of an array is being referenced.

There may be any number of expressions but each must evaluate to a floating point number. A <subscript list> as part of a DIM statement may not contain a reference to the array being dimensioned.

EXAMPLES:

```
X(10,20,20)

Y$(I,J)

COST(AMT(I),PRICE(I))
```

ELEMENT:

SQR (<expression>)

DESCRIPTION:

SQR returns the square root of the absolute value of the <expression>. The argument must evaluate to a floating point number.

EXAMPLES:

```
SQR (Y)

SQR(Xt2 + Yt2)
```

ELEMENT:

TAB predefined function

SYNOPSIS:

TAB (<expression>)

DESCRIPTION:

The TAB function positions the output buffer pointer to the position specified by the integer value of the <expression> rounded to the nearest integer modulo 73. If the value of the rounded expression is less than or equal to the current print position, the print buffer is dumped and the buffer pointer is set as described above.

The TAB function may occur only in PRINT statements.

EXAMPLES:

```
TAB(10)

TAB(I + 1)
```

ELEMENT:

STOP statement

SYNOPSIS:

[<line number>] STOP

DESCRIPTION:

Upon encountering a <STOP statement> program execution terminates and all open files are closed. The print buffer is emptied and control returns to the host system. Any number of STOP statements may appear in a program.

A STOP statement is appended to all programs by the compiler.

EXAMPLES:

```
10  STOP

    STOP
```

ELEMENT:

TAN predefined function

SYNOPSIS:

TAN (<expression>)

DESCRIPTION:

TAN is a function which returns the tangent of the expression. The argument must be in radians.

An error occurs if the <expression> is a multiple of pi/2 radians.

EXAMPLES:

```
10  TAN(A)

    TAN(X - 3*COS(Y))
```

ELEMENT:

VAL predefined function

SYNOPSIS:

VAL (<expression>)

DESCRIPTION:

The VAL function converts the number in ASCII passed as a parameter into a floating point number. The <expression> must evaluate to a string.

Conversion continues until a character is encountered that is not part of a valid number or until the end of the string is encountered.

EXAMPLES:

```
VAL(A$)

VAL("3.789" + "E-07" + "THIS IS IGNORED")
```

ELEMENT:

<variable>

SYNOPSIS:

<identifier> [(<subscript list>)]

DESCRIPTION:

A <variable> in BASIC-E may either represent a floating point number or a string depending on the type of the <identifier>. Subscripted variables must appear in a DIM statement before being used as a <variable>.

EXAMPLES:

```
X

Y$(5,10)


ABS.AMT(X(I),Y(I),S(I-1))
```

APPENDIX II - COMPILER ERROR MESSAGES

CE	could not close file.
DE	Disk error.
DF	Could not create INT file; disk or directory is full.
DL	Duplicate labels or syncronization error.
DP	Identifier in DIM statement previously defined.
FC	Identifier in FILE statement previously defined.
FD	Predefined function name previously defined.
FI	FOR loop index is not a simple floating point variable.
FN	Incorrect number of parameters in function reference.
FP	Invalid parameter type in function reference.
FU	Function is undefined.
IC	Invalid character in BASIC statement.
IE	Expression in IF statement is not of type floating-point.
IS	Subscripted variable not previously dimensioned.
IU	Array name used as simple variable.
MF	Expression is of type string where only floating point is allowed.
MM	Expression contains string and floating point variables in mixed mode expression.
NI	Identifier following NEXT does not match FOR statement index.
NP	No applicable production exists.
NS	No BAS file found.

NU	NEXT statement without corresponding FOR statement.
SN	Incorrect number of subscripts.
SO	Compiler stack overflow.
TO	Symbol table overflow.
VO	VARC overflow.

APPENDIX III - RUN-TIME MONITOR ERROR MESSAGES

AC	Null string passed as parameter to ASC function.
CE	Error closing a file.
DR	Disk read error (reading unwritten data in random access).
DW	Error writing to a file.
DZ	Division by zero.
EF	Eof on disk file.; no action specified.
ER	Exceeded record size on block file.
II	Invalid input from the console.
IR	Invalid record number in random access.
FU	Accessing an unopened file.
ME	Error attempting to create a file.
NE	Attempt to raise a number to a negative power.
NI	No INT file found in directory.
OD	Attempt to read past end of data area.
OE	Error attempting to open a file.
OI	Index in ON statement out of bounds.
RE	Attempt to read past end of record on blocked file.
RU	Unblocked file used with random access.
SB	Array subscript out of bounds.
SL	String length exceeds 255.
SS	Second parameter of MID$ is negative.
TZ	Attempt to evaluate tangent of pi over two.

APPENDIX IV - OPERATING INSTRUCTIONS FOR BASIC-E

The BASIC-E programs are written to operate with the CP/M Floppy Disk Operating System. Operation with a different system will require modification to the input/output routines in the compiler and run-time monitor. Execution of a program using BASIC-E consists of three steps. First the source program must be created on disk. Next the program is compiled by executing the BASIC-E compiler with the name of the source program provided as a parameter. Finally the intermediate (INT) file created by the compiler may be interpreted by executing the run-time monitor, again using the the source program name as a parameter.

Creation of the source program will normally be accomplished using CP/M's text editor, and must have a file type BAS. The BASIC-E statements are free form with the restriction that when a statement is not completed on a single line, a continuation character (\) must be the last character on the line. Spaces may precede statements and any number of spaces may appear wherever one space is permitted. Line numbers need only be used on statements to which control is passed. The line numbers do not have to be in ascending order. Using identifiers longer than two characters and indenting statements to enhance readability does

not affect the size of the object file created by the compiler.

The first statement of a source program may be used to specify certain compiler options. If present, this statement must begin with a dollar sign ($) in column one and be followed by the letter or letters indicating the options which are desired. The letters may be separated by any number of blanks. Invalid letters or characters are ignored. Appendix D lists valid compiler options, and their initial settings. Toggle A is used for compiler debugging. Toggle B supresses listing of the source program except for statements with errors. Toggle C compiles the program but does not create a INT file. Normally the BASIC-E compiler converts all letters appearing in identifiers or reserved words to uppercase. If toggle D is set this conversion is not performed. Letters appearing in strings are never converted to uppercase. Toggle E causes code to be generated by the compiler so that, upon detection of a run-time error, the source statement line which was being executed at the time the error occured is listed along with the error message.

The BASIC-E compiler is invoked as follows:

BASIC <program name>

The compiler begins execution by opening the source file specified as a parameter and compiles each BASIC-E statement producing an object file in the BASIC-E machine language with the same name as the source program but of type "INT".

The source program may be listed on the output device with any error messages following each line of the program. If no errors occur during compilation, the object file may be executed by the run time monitor by typing the command:

RUN <program name>

```
8080 PLM1 VERS 4.1

00001  1
00002  1   100H:               /* LOAD POINT FOR COMPILER  */
00003  1
00004  1
00005  1           /*
00006  1            ************************************************************
00007  1            *                                                          *
00008  1            *                      NBASIC COMPILER                     *
00009  1            *                                                          *
00010  1            *              U. S. NAVY POSTGRADUATE SCHOOL              *
00011  1            *                   MONTEREY, CALIFORNIA                   *
00012  1            *                                                          *
00013  1            *              WRITTEN BY GORDON EUBANKS, JR.              *
00014  1            *                                                          *
00015  1            *                      CPM VERSION 1.2                     *
00016  1            *                                                          *
00017  1            *                       NOVEMBER 1976                      *
00018  1            *                                                          *
00019  1            ************************************************************
00020  1           */
00021  1
00022  1           /*
00023  1            ************************************************************
00024  1            *                                                          *
00025  1            *     THE NBASIC COMPILER IS DIVIDED INTO THE FOLLOW-      *
00026  1            *     ING MAJOR SECTIONS:                                  *
00027  1            *           (1) GLOBAL DECLERATIONS AND LITERAL            *
00028  1            *               DEFINITIONS                                *
00029  1            *           (2) SYSTEM INPUT OUTPUT ROUTINES AND           *
00030  1            *               ASSOCIATED VARIABLE DECLERATIONS           *
00031  1            *           (3) SCANNER                                    *
00032  1            *           (4) SYMBOL TABLE ROUTINES                      *
00033  1            *           (5) PARSER AND CODE GENERATION                 *
00034  1            *                                                          *
00035  1            *     NBASIC REQUIRES A SOURCE PROGRAM AVAILABLE ON        *
00036  1            *     AN INPUT DEVICE AND WILL WRITE A BINARY OUTPUT       *
00037  1            *     FILE WHICH MAY BE EXECUTED BY THE RUN TIME           *
00038  1            *     MONITOR.  THE SOURCE MUST BE READ TWICE.             *
00039  1            *     THE NORMAL OUTPUT DEVICE IS THE CONSOLE.             *
00040  1            *                                                          *
00041  1            *     MODIFICATION OF THE COMPILER FOR OTHER OPERATING     *
00042  1            *     SYSTEMS WILL REQUIRE MODIFICATIONS TO SECTION        *
00043  1            *     (2) AND IN SECTION 1 REDEFINITION OF LITERALS IN     *
00044  1            *     SECTIONS  SYSTEM PARAMETERS WHICH MAY REQUIRE        *
00045  1            *     MODIFICATION BY USERS  AND  EXTERNAL ENTRY           *
00046  1            *     POINTS .  OTHER CHANGES SHOULD NOT BE REQUIRED       *
00047  1            *                                                          *
00048  1            ************************************************************
00049  1           */
00050  1
00051  1
00052  1           /*
00053  1            ************************************************************
00054  1            *                                                          *
00055  1            *          ****     SECTION  1     ****                  *
00056  1            *                                                          *
00057  1            ************************************************************
00058  1           */
00059  1           /*
00060  1            ************************************************************
00061  1            *                                                          *
00062  1            *                    GLOBAL LITERALS                       *
00063  1            *                                                          *
00064  1            ************************************************************
00065  1           */
00066  1
00067  1   DECLARE
00068  1           LIT     LITERALLY       'LITERALLY',
00069  1           TRUE            LIT     '1',
00070  1           FALSE           LIT     '0',
00071  1           FOREVER         LIT     'WHILE TRUE',
00072  1           INDEXSIZE       LIT     'ADDRESS',
00073  1           STATESIZE       LIT     'ADDRESS',
00074  1           LF              LIT     '0AH',
00075  1           QUESTIONMARK    LIT     '3FH',
00076  1           POUNDSIGN       LIT     '23H',
00077  1           UPARROW         LIT     '5EH',
00078  1           TAB             LIT     '09H',
00079  1           COLIN           LIT     '3AH',
00080  1           ASTRICK         LIT     '2AH',
00081  1           PERCENT         LIT     '25H';
00082  1
00083  1           /*
00084  1            ************************************************************
00085  1            *                                                          *
00086  1            *                  EXTERNAL ENTRY POINTS                   *
00087  1            *    THESE ENTRY POINTS ALLOW INTERFACEING WITH CP/M       *
00088  1            *                                                          *
00089  1            ************************************************************
00090  1           */
00091  1
```

```
00092  1  DECLARE
00093  1          BDOS    LIT             '05H',  /* ENTRY POINT TO CP/M */
00094  1          BOOT    LIT             '0H',   /* RETURN TO SYSTEM */
00095  1          STARTBDOS ADDRESS INITIAL(6H),  /* ADDR OF PTR TO TOP OF BDOS */
00096  1          MAX BASED STARTBDOS ADDRESS;  /* MAX USEABLE ADDRESS */
00097  1
00098  1          /*
00099  1           ************************************************************
00100  1           *                                                          *
00101  1           *              SYSTEM PARAMETERS WHICH MAY                 *
00102  1           *              REQUIRE MODIFICATION BY USERS               *
00103  1           *                                                          *
00104  1           ************************************************************
00105  1          */
00106  1
00107  1  DECLARE
00108  1          IDENTSIZE       LIT     '32',   /* MAX IDENTIFIER SIZE + 1 */
00109  1          VARCSIZE        LIT     '100',  /* SIZE OF VARC STACK */
00110  1          PSTACKSIZE      LIT     '14',   /* SIZE OF PARSE STACKS */
00111  1          EOLCHAR         LIT     '0DH',  /* END OF SOURCE LINE INDICATOR */
00112  1          EOFFILLER       LIT     '1AH', /* PAD CHAR FOR LAST REC ON FILE */
00113  1          SOURCERECSIZE   LIT     '128',  /* SIZE OF SOURCE FILE REC */
00114  1                      /* NOTE: THIS IS MAX SIZE OF SOURCE FILE RECORDS
00115  1                               IF SOURCE FILE CONSISTS OF VAR LNG REC */
00116  1          INTRECSIZE      LIT     '128',  /* INTERMEDIATE FILE REC SIZE */
00117  1          CONBUFFSIZE     LIT     '82',   /* SIZE OF CONSOLE BUFFER */
00118  1          HASHTBLSIZE     LIT     '64',   /* SIZE OF HASHTABLE */
00119  1          HASHMASK        LIT     '63',   /* HASHTBLSIZE - 1 */
00120  1          STRINGDELIM     LIT     '22H',  /* CHAR USED TO DELIM STRINGS */
00121  1          CONTCHAR        LIT     '5CH',  /* CONTINUATION CHARACTER */
00122  1          MAXONCOUNT      LIT     '15';   /* MAX NUMBER ON STATEMENTS */
00123  1
00124  1
00125  1          /*
00126  1           ************************************************************
00127  1           *                                                          *
00128  1           *                    GLOBAL VARIABLES                      *
00129  1           *                                                          *
00130  1           ************************************************************
00131  1          */
00132  1  DECLARE
00133  1          PASS1           BYTE INITIAL(TRUE),  /* PASS1 FLAG */
00134  1          PASS2           BYTE INITIAL(FALSE), /* PASS2 FLAG */
00135  1                  /*
00136  1                     COPILER TOGGLES
00137  1                  */
00138  1          LISTPROD        BYTE INITIAL(FALSE),
00139  1          LISTSOURCE      BYTE INITIAL(FALSE),
00140  1          DEBUGLN         BYTE INITIAL(FALSE),
00141  1          LOWERTOUPPER    BYTE INITIAL(TRUE),
00142  1          NOINTFILE       BYTE INITIAL(FALSE),
00143  1          ERRSET          BYTE INITIAL(FALSE),
00144  1          ERRORCOUNT      BYTE INITIAL(0),
00145  1          ULERRORFLAG     BYTE INITIAL(FALSE),
00146  1          COMPILING       BYTE,
00147  1          CODESIZE        ADDRESS,  /* USED TO COUNT SIZE OF CODE AREA */
00148  1          PRTCT           ADDRESS,  /* USED TO COUNT NUMBER OF PRT ENTRIES */
00149  1          FDACT           ADDRESS,  /* USED TO COUNT NUMBER OF FDA ENTRIES */
00150  1          DATACT          ADDRESS,  /* USED TO COUNT SIZE OF DATA AREA */
00151  1
00152  1                  /* VARIABLES USED DURING FOR LOOP CODE GENERATION */
00153  1
00154  1          FORSTMT         BYTE,
00155  1          NEXTSTMTPTR     ADDRESS,
00156  1          NEXTADDRESS     BASED NEXTSTMTPTR(4) ADDRESS,
00157  1          NEXTBYTE BASED NEXTSTMTPTR BYTE,
00158  1          FORCCUNT BYTE INITIAL(0),
00159  1
00160  1                 /*  FLAGS USED DURING CODE GENERATION */
00161  1
00162  1          RANDOMFILE      BYTE,
00163  1          FILEIO          BYTE,
00164  1          INPUTSTMT       BYTE,
00165  1          GOSUBSTMT       BYTE,
00166  1
00167  1    /*    THE FOLLOWING GLOBAL VARIABLES ARE USED BY THE SCANNER      */
00168  1
00169  1          TOKEN       BYTE,  /* TYPE OF TOKEN JUST SCANNED */
00170  1          SUBTYPE     BYTE,  /* SUBTYPE OF CURRENT TOKEN */
00171  1          FUNCCP      BYTE,  /* IF TOKEN FUNC THEN THIS IS FUNC NUMBER */
00172  1          HASHCODE    BYTE,  /* HASH VALUE OF CURRENT TOKEN */
00173  1          NEXTCHAR    BYTE,  /* CURRENT CHARACTER FROM GETCHAR */
00174  1          ACCUM(IDENTSIZE)     BYTE,  /* HOLDS CURRENT TOKEN */
00175  1          CONT        BYTE,  /* INDICATES ACCUM WAS FULL, STILL MORE */
00176  1
00177  1    /*    SYMBOL TABLE GLOBAL VARIABLES  */
00178  1
00179  1          BASE            ADDRESS,  /* BASE OF CURRENT ENTRY */
00180  1          HASHTABLE(HASHTBLSIZE) ADDRESS,
00181  1          SBTBLTOP        ADDRESS,  /* CURRENT TOP OF SYMBOL TABLE */
00182  1          FORADDRESS BASED SBTBLTOP(4) ADDRESS, /* FOR STATEMENT INFO */
00183  1          SBTBL           ADDRESS,
00184  1          PTR BASED BASE BYTE,  /* FIRST BYTE OF ENTRY */
00185  1          APTRADDR        ADDRESS,  /* UTILITY VARIABLE TO ACCESS TABLE */
00186  1          BYTEPTR BASED APTRADDR BYTE,
00187  1          ADDRPTR BASED APTRADDR ADDRESS,
00188  1          PRINTNAME       ADDRESS,  /* SET PRIOR TO LOOKUP OR ENTER */
00189  1          SYMHASH         BYTE,  /* ALSO SET PRIOR TO LOOKUP OR ENTER */
```

```
00190  1
00191  1              /*
00192  1               ************************************************
00193  1               *                                              *
00194  1               *  THE FOLLOWING LITERAL DEFINITIONS ESTABLISH  *
00195  1               *  MEMONIC NAMES FOR THE TOKENS WHICH ARE THE   *
00196  1               *  OUTPUT OF THE LALR PARSER PROGRAM.           *
00197  1               *                                              *
00198  1               ************************************************
00199  1              */
C0200  1            POUND  LIT '12', LPARN LIT '02', RPARN  LIT '05',
C0201  1            ASTRK LIT '04', TPLUS LIT '03', TMINUS LIT '07',
00202  1            LESST LIT '01', TCOLIN LIT '11', SCOLN  LIT '06',
00203  1            EXPON LIT '14', EQUAL LIT '13', GTRT   LIT '10',
00204  1            TDATA LIT '99', TAND  LIT '24', TCR    LIT '23',
00205  1            TELSE LIT '34', TDEF  LIT '25', TDIM   LIT '26',
00206  1            TFOR  LIT '28', TEND  LIT '27', TFILE  LIT '35',
00207  1            TIF   LIT '17', TGOSB LIT '43', TGOTO  LIT '36',
00208  1            TNEXT LIT '37', TINPT LIT '44', TLET   LIT '29',
C0209  1            SLASH LIT '08', TNOT  LIT '30', TON    LIT '20',
C0210  1            TOR   LIT '21', TPRNT LIT '45', TREAD  LIT '38',
00211  1            TREST LIT '48', TRETN LIT '46', TSTEP  LIT '39',
00212  1            TSTOP LIT '40', TTHEN LIT '41', TTO    LIT '22',
00213  1            FUNCT LIT '53', TGEQ  LIT '15', TSUB   LIT '32',
00214  1            TLEQ  LIT '18', COMMA LIT '09', TGO    LIT '16',
00215  1            TNE   LIT '19', TCLOS LIT '42', TXOR   LIT '33',
00216  1            TOUT  LIT '31', TIRN LIT '51',  STRING LIT '50',
00217  1            IDENTIFIER LIT '52',            FLOATPT LIT '49',
00218  1            UDFUNCT LIT '54',               TREM LIT '0';
00219  1
C0220  1            /*
00221  1             ************************************************
00222  1             *                                              *
00223  1             *                LALR PARSE TABLES             *
00224  1             *                  AND VARIABLES               *
00225  1             *                                              *
00226  1             ************************************************
00227  1            */
C0228  1    DECLARE MAXRNO LITERALLY '120',/* MAX READ COUNT */
C0229  1            MAXLNO LITERALLY '175',/* MAX LOOK COUNT */
00230  1            MAXPNO LITERALLY '189',/* MAX PUSH COUNT */
00231  1            MAXSNO LITERALLY '341',/* MAX STATE COUNT */
00232  1            STARTS LITERALLY '121';/* START STATE */
00233  1            PRODNO LITERALLY '152';/* NUMBER OF PRODUCTIONS */
00234  1    DECLARE READ1 DATA(0,49,10,13,2,49,50,52,53,54,49,13,22,32,2,3,7,27,30
00235  1         ,49,50,52,53,54,2,3,7,30,49,50,52,53,54,54,52,12,52,2,3,7,49,50,52
00236  1         ,53,54,12,52,49,49,50,2,3,7,12,30,49,50,52,53,54,2,2,2,5,5,9,49,4,8
00237  1         ,49,16,20,28,29,31,35,36,37,38,40,42,43,44,45,46,48,49,51,52,49,14,6
00238  1         ,22,13,52,5,52,9,23,9,21,33,41,16,21,33,36,43,9,21,33,5,9,21,33,5,21
C0239  1         ,33,5,9,21,33,5,9,21,33,5,9,21,33,21,33,39,21,33,41,5,21,33,6,21,33
C0240  1         ,9,6,9,16,17,20,25,26,27,23,29,31,35,36,37,38,40,42,43,44,45,46,48
00241  1         ,51,52,2,16,20,28,29,31,35,36,37,38,40,42,43,44,45,46,48,51,52,52,13
00242  1         ,24,11,34,9,2,2,3,7,10,13,15,18,19,3,7,9,0);
00243  1    DECLARE LOOK1 DATA(0,49,0,10,13,0,13,0,11,23,34,0,52,0,12,52,0,49,50,0,6
00244  1         ,9,11,23,34,0,2,0,2,0,9,0,4,8,0,4,8,0,4,8,0,4,8,0,4,8,0,11,23,34,0
00245  1         ,14,0,14,0,14,0,9,0,9,0,9,0,9,0,9,0,21,33,0,21,33,0,21,33,0,21,33,0
00246  1         ,21,33,39,0,21,33,0,21,33,0,21,33,0,23,0,21,33,0,21,33,0,9,0,9,0,6,9
00247  1         ,0,52,0,11,23,0,11,23,34,0,2,0,11,23,0,52,0,24,0,24,0,11,0,23,0,11,0
00248  1         ,9,0,2,0,1,3,7,10,13,15,18,19,0,3,7,0,9,0);
C0249  1    DECLARE APPLY1 DATA(0,0,0,0,55,105,0,19,0,0,32,47,0,0,3,4,12,14,16,17,20
C0250  1         ,21,22,26,27,34,36,38,40,98,100,102,103,114,116,0,0,46,0,28,0,33,0
00251  1         ,63,0,5,6,8,9,0,7,10,0,23,0,13,19,32,35,47,55,99,101,105,106,0,0,0,0
00252  1         ,0,39,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,11,0,0,0,0,0,0,0,99
00253  1         ,106,0,0,0,0,0,0,40,0,0,0,0,0,0,0,62,0,0,74,0,74,0,0,0,0,0,0,0,0,0);
00254  1    DECLARE READ2(205) ADDRESS INITIAL
00255  1                       (0,191,264,260,3,255,256,129,254,253,326,258,329,331,3
00256  1         ,5,8,31,33,255,256,129,254,253,3,5,8,33,255,256,129,254,253,279,42
00257  1         ,21,129,3,5,8,255,256,129,254,253,20,129,273,255,256,3,5,8,20,33,255
C0258  1         ,256,129,254,253,247,294,4,335,280,283,320,7,10,327,24,26,268,32,34
C0259  1         ,285,328,125,126,338,38,330,127,128,337,340,275,341,129,325,23,302
C0260  1         ,27,220,130,17,131,13,190,14,223,224,277,24,223,224,328,330,12,223
00261  1         ,224,246,248,223,224,244,223,224,249,252,223,224,293,295,223,224,316
00262  1         ,16,223,224,223,224,36,223,224,37,288,223,224,317,223,224,15,318,319
C0263  1         ,24,25,26,29,30,339,268,32,34,285,328,125,126,338,38,330,127,128,337
00264  1         ,340,341,129,251,24,26,268,32,34,285,328,125,126,338,38,330,127,128
00265  1         ,337,340,341,129,45,22,28,124,276,286,282,122,6,9,123,257,259,261
00266  1         ,265,6,9,11,0);
00267  1    DECLARE LOOK2(151) ADDRESS INITIAL
00268  1                       (0,1,176,2,2,265,18,262,177,177,177,19,334,333,35,35
C0269  1         ,178,39,39,179,180,180,180,180,180,40,41,245,43,181,44,332,49,49,231
C0270  1         ,50,50,234,51,51,235,52,52,232,53,53,233,182,182,182,55,57,236,58
00271  1         ,237,59,238,66,308,68,300,69,299,70,301,72,296,76,76,297,77,77,309
00272  1         ,78,78,219,84,84,312,85,85,35,183,87,87,336,88,88,298,89,89,310,278
00273  1         ,91,93,93,313,94,94,269,95,321,96,322,97,97,184,99,185,186,186,101
00274  1         ,314,314,314,102,104,250,187,187,105,106,188,109,221,110,222,111,193
00275  1         ,274,112,113,272,115,284,117,189,118,118,118,118,118,118,118,118,229
00276  1         ,119,119,230,120,290);
00277  1    DECLARE APPLY2(128) ADDRESS INITIAL
00278  1                       (0,0,161,71,169,170,168,199,198,200,218,267,201,98,80
00279  1         ,90,151,152,92,155,83,86,154,74,150,75,156,146,147,148,149,153,82,79
C0280  1         ,81,73,46,167,166,226,225,228,227,174,173,133,135,134,136,132,139
00281  1         ,140,138,240,239,305,64,64,304,64,64,304,64,64,304,241,114,243,116
00282  1         ,163,60,242,63,202,61,47,266,194,271,164,137,197,172,108,107,204,65
00283  1         ,171,287,196,175,292,291,103,205,145,206,210,165,143,144,142,207,159
00284  1         ,141,307,100,160,162,208,213,56,62,158,157,209,323,48,324,54,203,67
00285  1         ,216,212,211,195,214,215);
```

```
00286  1    DECLARE INDEX1(341) ADDRESS INITIAL
00287  1                        (0,1,2,24,24,4,4,4,4,4,4,34,24,36,24,10,24,24,11,168
C0288  1         ,24,24,24,4,12,14,24,24,24,33,34,35,36,37,24,45,24,47,24,48,50,60,61
C0289  1         ,62,63,64,24,36,66,67,67,67,67,67,69,70,89,90,90,90,91,92,89,37,93
C0290  1         ,94,95,96,97,97,97,98,99,100,103,108,100,100,100,111,115,118,122,126
00291  1         ,100,130,133,100,100,100,136,100,139,100,100,142,142,143,24,36,24
00292  1         ,145,24,24,167,168,36,186,187,188,188,189,189,189,24,191,24,192,193
00293  1         ,201,203,1,3,6,8,12,14,17,20,26,28,30,32,35,38,41,44,47,51,53,55,57
00294  1         ,59,61,63,65,67,70,73,76,79,83,86,89,92,94,97,100,102,104,107,109
00295  1         ,112,116,118,121,123,125,127,129,131,133,135,137,146,149,192,217,306
00296  1         ,303,311,289,217,270,315,306,217,217,306,281,1,2,2,3,3,3,3,3,4,4,7,7
00297  1         ,7,7,7,7,7,7,7,7,7,7,7,7,7,7,9,10,13,14,14,36,36,37,37,39,39,41
C0298  1         ,41,43,43,43,43,43,45,45,45,50,50,53,53,53,53,55,55,66,66,67,67,68
00299  1         ,68,69,69,70,70,72,72,72,72,72,72,72,72,72,73,74,75,76,76,77,77,77
C0300  1         ,78,78,79,80,81,82,83,83,84,84,85,86,86,87,88,88,89,90,90,90,91,93,93
00301  1         ,94,95,95,96,96,97,98,98,99,99,99,102,102,103,103,103,104,104,105
00302  1         ,105,106,106,108,108,109,110,110,111,112,113,113,115,116,116,118,118
00303  1         ,120,120,121,121,122,123,124,125,126,127);
00304  1    DECLARE INDEX2 DATA(0,1,2,9,9,6,6,6,6,6,6,1,9,1,9,1,9,9,1,18,9,9,9,6,2
00305  1         ,10,9,9,9,1,1,1,1,8,9,2,9,1,9,2,10,1,1,1,1,2,9,1,1,2,2,2,2,2,1,19,1
00306  1         ,1,1,1,1,1,1,8,1,1,1,1,1,1,1,1,1,3,5,3,2,2,2,4,3,4,4,4,2,3,3,2,2,2,3
00307  1         ,2,3,2,2,1,1,2,9,1,9,22,9,9,1,18,1,1,1,1,1,1,2,1,9,1,9,1,8,2,1,2,3,2
00308  1         ,4,2,3,3,6,2,2,2,2,3,3,3,3,3,4,2,2,2,2,2,2,2,2,3,3,3,3,4,3,3,3,2,3,3,2
C0309  1         ,2,3,2,3,4,2,3,2,2,2,2,2,2,2,2,9,3,2,1,19,35,39,40,43,55,85,97,99
C0310  1         ,101,105,108,117,2,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
00311  1         ,0,0,1,1,1,0,2,0,0,0,2,0,1,0,2,0,2,2,1,1,0,2,2,0,2,0,0,0,2,0,2,1,2,2
00312  1         ,0,1,2,0,0,0,0,0,1,0,1,0,0,0,1,0,3,1,0,1,0,0,1,5,1,1,2,2,3,1,2,0,0,2
00313  1         ,1,0,2,1,2,0,1,0,2,2,1,2,1,0,2,2,1,2,1,0,0,2,0,2,2,0,2,0,0,2,0,0,2,4
00314  1         ,0,0,1,1,1,2,2,0,2,1,0,1,0,1,1,0,0,2,3,0,0,0,0,0);
00315  1             /*
00316  1              ************************************************************
00317  1              *                                                          *
00318  1              *        ****        SECTION    2        ****             *
00319  1              *                                                          *
C0320  1              *       SYSTEM DEPENDENT ROUTINES AND VARIABLES            *
00321  1              *                                                          *
00322  1              *             THE FOLLOWING ROUTINES ARE USED              *
00323  1              *             BY THE COMPILER TO ACCESS DISK               *
00324  1              *             FILES AND THE CONSOLE.  THESE                *
00325  1              *             ROUTINES ASSUME THE USE OF THE               *
00326  1              *             CP/M DISK OPERATING SYSTEM.                  *
00327  1              *                                                          *
00328  1              *       THE FCB'S ARE USED BY THE SYSTEM TO MAINTAIN       *
C0329  1              *       INFORMATION ON OPEN FILES.  THEY ARE ONLY USED     *
00330  1              *       BY PROCEDURES IN THIS SECTION.  THE BUFFERS        *
00331  1              *       AND POINTERS TO THE BUFFERS ARE USED BY THE        *
00332  1              *       REMAINDER OF THE PROGRAM BUT THEIR SIZE MAY        *
00333  1              *       BE VARIED TO SUIT THE DISK SYSTEM BEING USED       *
00334  1              *                                                          *
00335  1              ************************************************************
00336  1             */
00337  1
00338  1    DECLARE
C0339  1            RFCBADDR    ADDRESS INITIAL(5CH),
C0340  1              /* NOTE: CP/M PROVIES 5CH AS FCB AREA AND 80H AS A
00341  1                 BUFFER FOR PROGRAM USE */
00342  1            RFCB BASED RFCBADDR(33)     BYTE,  /* SOURCE FCB */
00343  1            WFCB(33)                    BYTE  /* INTERMEDIATE FILE FCB */
00344  1                                        INITIAL(0,'          ','INT',0,0,0,0),
00345  1            SBLOC                       ADDRESS INITIAL(80H),
00346  1            SOURCEBUFF BASED SBLOC(SOURCERECSIZE)BYTE,  /* SOURCE BUFFER */
00347  1            SOURCEPTR       BYTE INITIAL(SOURCERECSIZE), /* BUFFER INDEX */
00348  1            CURSOURCERECSIZE BYTE INITIAL(SOURCERECSIZE),
00349  1            DISKOUTBUFF(INTRECSIZE)       BYTE,
C0350  1            BUFFPTR                 BYTE INITIAL(255), /* BUFFER INDEX */
00351  1            LINEBUFF(CONBUFFSIZE)         BYTE,  /* CONSOLE OUT BUFFER */
00352  1            LINEPTR                 BYTE INITIAL(0), /* BUFFER INDEX */
00353  1            LINENO                        ADDRESS, /* CURRENT LINE NUMBER */
00354  1             SEPARATOR              BYTE INITIAL(COLIN);
00355  1
C0356  1           /*  MON1, MON2, AND MON3 ARE CP/M CALLS TO PERFORM SYSTEM
00357  1               PROVIDED FUNCTIONS.  THE LITERAL DECLARATIONS
00358  1               BELOW DEFINE THESE FUNCTIONS.  THE SECOND PARAMETER IN
C0359  1               MON1 AND MON2 IS A DATA ITEM SUCH AS A BUFFER LOCATION
C0360  1           */
00361  1
00362  1               DECLARE
00363  1                       PCHAR    LIT     '2',  /* CHAR TO CONSOLE */
00364  1                       PBUFF    LIT     '9',  /* BUFFER TO CONSOLE */
00365  1                       RCHAR    LIT     '1',  /* CHAR FROM CONSOLE */
C0366  1                       RBUFF    LIT    '10',  /* BUFFER FROM CONSOLE */
00367  1                       OFILE    LIT    '15',  /* OPEN FILE */
00368  1                       CFILE    LIT    '16',   /* CLOSE FILE */
C0369  1                       DFILE    LIT    '19',  /* DELETE FILE */
00370  1                       RFILE    LIT    '20',  /* READ FILE */
00371  1                       WFILE    LIT    '21',  /* WRITE FILE */
00372  1                       MFILE    LIT    '22',  /* MAKE FILE */
00373  1                       SDMA     LIT    '26',  /* SET DMA */
00374  1                       FILEERR  LIT   '255',  /* ERROR RTN CODE */
C0375  1                       FILEEOF  LIT     '1';  /* EOF RTN CODE */
00376  1
00377  1    MON1: PROCEDURE(F,A);
00378  2          DECLARE F BYTE,
C0379  2          A ADDRESS;
C0380  2          GO TO BDOS;
00381  2    END MON1;
00382  1
```

```
00383  1    MON2: PROCEDURE (F,A) BYTE;
00384  2         DECLARE F BYTE, A ADDRESS;
00385  2         GO TO BDOS;
00386  2         END MON2;
00387  1
00388  1
00389  1    MON3: PROCEDURE;
00390  2         /* USED TO RETURN TO THE SYSTEM */
00391  2        HALT;  /* FOR OMRON SYSTEMS */
00392  2        GOTO BOOT;  /* RETURN TO CP/M */
00393  2    END MON3;
00394  1
00395  1
00396  1    MOVE: PROCEDURE (SOURCE,DEST,COUNT);
00397  2        DECLARE
00398  2                SOURCE ADDRESS,
00399  2                DEST   ADDRESS,
00400  2                COUNT      BYTE,
00401  2                SCHAR BASED SOURCE BYTE,
00402  2                DCHAR BASED DEST   BYTE;
00403  2
00404  2        DO WHILE(COUNT := COUNT -1) <> 255;
00405  2             DCHAR = SCHAR;
00406  3             SOURCE = SOURCE + 1;
00407  3             DEST = DEST + 1;
00408  3             END;
00409  2        RETURN;
00410  2    END MOVE;
00411  1
00412  1    FILL: PROCEDURE (DEST,CHAR,COUNT);
00413  2          /* MOVE CHAR TO A N TIMES */
00414  2        DECLARE
00415  2                DEST         ADDRESS,
00416  2                CHAR         BYTE,
00417  2                COUNT        BYTE,
00418  2                DCHAR        BASED      DEST      BYTE;
00419  2         DO WHILE (COUNT := COUNT -1) <> 255;
00420  2              DCHAR = CHAR;
00421  3              DEST = DEST + 1;
00422  3              END;
00423  2        RETURN;
00424  2    END FILL;
00425  1
00426  1
00427  1    PRINTCHAR: PROCEDURE(CHAR);
00428  2          DECLARE CHAR BYTE;
00429  2        CALL MON1(PCHAR,CHAR);
00430  2    END PRINTCHAR;
00431  1
00432  1
00433  1    PRINT: PROCEDURE(A);
00434  2          DECLARE A ADDRESS;
00435  2           CALL MON1(PBUFF,A);
00436  2    END PRINT;
00437  1
00438  1
00439  1    DISKERR: PROCEDURE;
00440  2        CALL PRINT(.'DE    $');
00441  2        CALL MON3;  /* RETURN TO SYSTEM */
00442  2        RETURN;
00443  2    END DISKERR;
00444  1
00445  1    OPEN$SOURCEFILE: PROCEDURE;
00446  2        /*  SETS UP THE FCB FOR THE SOURCE PROGRAM
00447  2            WHICH MUST BE OF TYPE 'BAS' AND THEN OPENS
00448  2            THE FILE.  CP/M PUTS THE NAME USED AS A
00449  2            PARAMETER WHEN THE COMPILER IS EXECUTED, AT
00450  2            5CH.
00451  2        */
00452  2
00453  2        CALL MOVE(.'BAS',RFCBADDR+9,3);
00454  2        RFCB(32) = 0;
00455  2        IF MON2(OFILE,RFCBADDR) = FILEERR THEN
00456  2             DO;
00457  2                  CALL PRINT(.'NS $');
00458  3                  CALL MON3;  /* RETURN TO SYSTEM */
00459  3             END;
00460  2    END OPEN$SOURCEFILE;
00461  1
00462  1
00463  1    REWIND$SOURCE$FILE: PROCEDURE;
00464  2        /* CP/M DOES NOT REQUIRE ANY ACTION PRIOR TO REOPENING */
00465  2        RETURN;
00466  2    END REWIND$SOURCE$FILE;
00467  1
00468  1
00469  1    CLOSE$INT$FILE: PROCEDURE;
00470  2        IF MON2(CFILE,.WFCB) = FILEERR THEN
00471  2             CALL DISKERR;
00472  2    END CLOSE$INT$FILE;
00473  1
00474  1    SETUP$INT$FILE: PROCEDURE;
00475  2          /*  MAKES A NEW FILE */
00476  2        IF NOINTFILE THEN  /* ONLY MAKE FILE IF THIS TOGGLE IS OFF */
00477  2             RETURN;
00478  2        CALL MOVE(.RFCB,.WFCB,9);
00479  2        CALL MON1(DFILE,.WFCB);
00480  2        IF MON2(MFILE,.WFCB) = FILEERR THEN
```

```
00481  2            CALL DISKERR;
00482  2     END SETUP$INT$FILE;
00483  1
00484  1     READ$SOURCE$FILE: PROCEDURE BYTE;
00485  2         DECLARE DCNT BYTE;
00486  2         IF(DCNT := MON2(RFILE,RFCBADDR)) > FILEEOF THEN
00487  2             CALL DISKERR;
00488  2         RETURN DCNT; /* ZERO IF READ ELSE 1 IF EOF - ERRORS > 1 */
C0489  2     END READ$SOURCE$FILE;
C0490  1
00491  1
00492  1     WRITE$INT$FILE: PROCEDURE;
00493  2         IF NOINTFILE THEN
00494  2             RETURN;
00495  2         CALL MON1(SDMA,.DISKOUTBUFF);
00496  2         IF MON2(WFILE,.WFCB) <> 0 THEN
00497  2             CALL DISKERR;
C0498  2         CALL MON1(SDMA,80H); /* RESET DMA ADDRESS */
C0499  2     END WRITE$INT$FILE;
C0500  1
00501  1
00502  1     CRLF: PROCEDURE;
00503  2          CALL PRINTCHAR(EOLCHAR);
00504  2          CALL PRINTCHAR(LF);
00505  2         RETURN;
00506  2     END CRLF;
00507  1
00508  1
C0509  1     PRINT$DEC: PROCEDURE(VALUE);
00510  2         /*
00511  2             CONVERTS VALUE TO A DECIMAL NUMBER WHICH IS PRINTED
00512  2             ON THE CONSOLE.  USED FOR LINENUMBERING STATEMENTS
00513  2             AND TO PRINT PRODUCTIONS.
00514  2         */
00515  2         DECLARE
00516  2                 VALUE ADDRESS,
00517  2                 I     BYTE,
00518  2                 FLAG  BYTE,
C0519  2                 COUNT BYTE;
C0520  2         DECLARE DECIMAL(4) ADDRESS INITIAL(1000,100,10,1);
00521  2         FLAG = FALSE;
00522  2         DO I = 0 TO 3;
00523  2             COUNT = 30H;
00524  3             DO WHILE VALUE >= DECIMAL(I);
00525  3                 VALUE = VALUE - DECIMAL(I);
00526  4                 FLAG = TRUE;
00527  4                 COUNT = COUNT + 1;
00528  4             END;
C0529  3             IF FLAG OR (I >= 3) THEN
00530  3                 CALL PRINTCHAR(COUNT);
00531  3             ELSE
00532  3                 CALL PRINTCHAR(' ');
00533  3         END;
00534  2         RETURN;
00535  2     END PRINTDEC;
00536  1
00537  1
C0538  1     SETFLAGS: PROCEDURE;
00539  2         /*
C0540  2           RESET COMPILER FLAGS USED DURING PARSING
00541  2         */
00542  2           RANDOMFILE,FILEIO,
00543  2           INPUTSTMT, FORSTMT, GOSUBSTMT = FALSE;
00544  2         RETURN;
00545  2     END SETFLAGS;
00546  1
00547  1
00548  1             /*
00549  1              ************************************************************
C0550  1              *                                                          *
00551  1              *      THE FOLLOWING ROUTINE GENERATES THE INTERMEDIATE    *
00552  1              *      LANGUAGE FILE. EMIT IS THE ONLY ROUTINE TO          *
00553  1              *      ACTUALLY WRITE TO THE DISK.  GENERATE, EMITDAT,     *
00554  1              *      AND EMITCON CALL EMIT.                              *
00555  1              *                                                          *
00556  1              ************************************************************
00557  1             */
00558  1
C0559  1
C0560  1
00561  1     EMIT: PROCEDURE(OBJCODE);
00562  2         DECLARE OBJCODE BYTE;
00563  2         IF (BUFFPTR:=BUFFPTR + 1) >= INTRECSIZE THEN /* WRITE TO DISK */
00564  2             DO;
00565  2                 CALL WRITE$INT$FILE;
00566  3                 BUFFPTR = 0;
00567  3             END;
00568  2         DISKOUTBUFF(BUFFPTR) = OBJCODE;
C0569  2         RETURN;
C0570  2     END EMIT;
00571  1
00572  1
00573  1
00574  1
00575  1
00576  1
_00577  1
```

```
00578  1          /*
00579  1           ************************************************************
C0580  1           *                                                          *
00581  1           *            ***     SCANNER SECTION     ***             *
00582  1           *                                                          *
00583  1           ************************************************************
00584  1          */
00585  1
00586  1    CLEAR$LINE$BUFF: PROCEDURE;
00587  2        CALL FILL(.LINEBUFF,' ',CONBUFFSIZE);
00588  2    END CLEAR$LINE$BUFF;
00589  1
C0590  1
00591  1    LIST$LINE: PROCEDURE(LENGTH);
00592  2        DECLARE
00593  2                LENGTH BYTE,
00594  2                I      BYTE;
00595  2        CALL PRINT$DEC(LINENO);
00596  2        CALL PRINT$CHAR(SEPARATOR);
00597  2        CALL PRINT$CHAR(' ');
00598  2        DO I = 0 TO LENGTH;
C0599  2            CALL PRINTCHAR(LINEBUFF(I));
C0600  3            END;
00601  2        CALL CRLF;
00602  2        CALL CLEAR$LINE$BUFF;
00603  2        SEPARATOR = COLIN;
00604  2        RETURN;
00605  2    END LIST$LINE;
00606  1
00607  1          /*
00608  1           ************************************************************
00609  1           *                                                          *
00610  1           *    GETCHAR SETS THE GLOBAL VARIABLE NEXTCHAR TO THE      *
00611  1           *    NEXT SOURCEFILE CHARACTER AND RETURNS NEXTCHAR TO     *
00612  1           *    THE CALLING ROUTINE.                                  *
00613  1           *                                                          *
00614  1           *    TABS ARE REPLACED WITH A BLANK AND IF EITHER          *
00615  1           *    LISTSOURCE IS TRUE OR AN ERROR HAS OCCURED LINES      *
00616  1           *    ARE OUTPUT TO THE CONSOLE.                            *
00617  1           *                                                          *
00618  1           ************************************************************
C0619  1          */
C0620  1
00621  1    GETCHAR: PROCEDURE BYTE;
00622  2        DECLARE ADDEND DATA ('END',EOLCHAR,LF); /*TO ADD END IF LEFT OFF */
00623  2        NEXT$SOURCE$CHAR: PROCEDURE BYTE;
00624  3            RETURN SOURCEBUFF(SOURCEPTR);
00625  3        END NEXT$SOURCE$CHAR;
00626  2
C0627  2        CHECKFILE: PROCEDURE BYTE;
C0628  3            /*
00629  3                CHECKFILE MAINTAINS THE SOURCE BUFFER FULL AND
C0630  3                CHECKS FOR END OF FILE ON THE SOURCE FILE.
00631  3                IF A LINE FEED IS FOUND IT IS SKIPPED.
00632  3                IF END OF FILE IS DETECTED THEN TRUE IS RETURNED
00633  3                ELSE FALSE IS RETURNED.
00634  3            */
00635  3            DO FOREVER;  /* ALLOW US TO SKIP LINE FEEDS */
00636  3                IF (SOURCEPTR := SOURCEPTR + 1) >= CURSOURCERECSIZE THEN
00637  4                    DO;
00638  4                        SOURCEPTR = 0;
00639  5                        IF READ$SOURCE$FILE = FILEEOF THEN
C0640  5                            RETURN TRUE;
00641  5                    END;
00642  4                IF (NEXTCHAR := NEXT$SOURCE$CHAR) <> LF THEN
00643  4                    RETURN FALSE;
00644  4            END;  /* OF DO FOREVER */
00645  3        END CHECKFILE;
00646  2
00647  2        IF CHECKFILE OR (NEXTCHAR = EOFFILLER)  THEN
00648  2                    DO; /* EOF REACHED */
00649  2                        CALL MOVE(.ADDEND,SBLOC,5);
C0650  3                        SOURCEPTR = 0;
00651  3                        NEXTCHAR = NEXT$SOURCE$CHAR;
00652  3                    END;
00653  2            LINEBUFF(LINEPTR := LINEPTR + 1) = NEXTCHAR;  /* OUTPUT LINE */
00654  2            IF NEXTCHAR = EOLCHAR THEN
00655  2                DO;
C0656  2                    LINENO = LINENO + 1;
00657  3                    IF LISTSOURCE OR ERRSET THEN
00658  3                        CALL LISTLINE(LINEPTR - 1);  /* NOT EOLCHAR */
C0659  3                    LINEPTR = 0;
C0660  3                END;
00661  2        IF NEXTCHAR = TAB THEN
00662  2            NEXTCHAR = ' ';  /* ONLY NEED REPLACE WITH 1 BLANK  */
00663  2        RETURN NEXTCHAR;
00664  2    END GETCHAR;
00665  1
00666  1
C0667  1    GETNOBLANK: PROCEDURE;
00668  2        DO WHILE((GETCHAR = ' ') OR (NEXTCHAR = EOFFILLER));
00669  2            END;
C0670  2        RETURN;
00671  2    END GETNOBLANK;
00672  1
00673  1
00674  1
```

```
00675  1
00676  1    CHECK$CONTINUATION: PROCEDURE;
00677  2        /*
00678  2            CHECK FOR CONTINUATION CHAR. IF FOUND SET NEXTCHAR
00679  2            TO FIRST CHARACTER ON NEXT LINE.  IT THEN LOOKS TO
00680  2            THE PARSER AS IF IT WAS ALL ONE LINE.
00681  2        */
00682  2        IF NEXTCHAR = CONTCHAR THEN
00683  2            DO;
00684  2                DO WHILE GETCHAR <> EOLCHAR;
00685  3                    END;
00686  3                CALL GETNOBLANK;
00687  3            END;
00688  2        RETURN;
00689  2    END CHECK$CONTINUATION;
00690  1
00691  1
00692  1            /*
00693  1             ***************************************************************
00694  1             *                                                             *
00695  1             *     ERROR IS THE COMPILER ERROR HANDLING ROUTINE            *
00696  1             *     IF AN ERROR IS DETECTED WHILE PARSING A STATEMENT       *
00697  1             *     THE REMAINDER OF THE STATEMENT IS SKIPPED AND THE       *
00698  1             *     STATEMENT IS WRITTEN ON THE CONSOLE FOLLOWED BY A       *
00699  1             *     TWO LETTER DISCRIPTION OF THE ERROR. AN UP ARROW        *
00700  1             *     INDICATES WHERE IN THE LINE THE ERROR WAS DETECTED      *
00701  1             *     THE PARSER IS RESET AND COMPILATION CONTINUES WITH      *
00702  1             *     THE NEXT STATEMENT.                                     *
00703  1             *                                                             *
00704  1             ***************************************************************
00705  1            */
00706  1
00707  1    ERROR: PROCEDURE(ERRCODE);
00708  2            DECLARE
00709  2                    ERRCODE ADDRESS,
00710  2                    POINTER BYTE;
00711  2        POINTER = LINEPTR + 2;
00712  2        IF PASS2 THEN
00713  2            ERRSET = TRUE;  /* SO SOURCE LINE WILL BE LISTED */
00714  2        IF TOKEN <> TCR THEN
00715  2            DO WHILE NEXTCHAR <> EOLCHAR;  /* SKIP REMAINDER OF LINE */
00716  2                CALL CHECK$CONTINUATION;
00717  3                NEXTCHAR = GETCHAR;
00718  3                END;
00719  2        IF PASS2 THEN
00720  2            DO;         /* PRINT ERROR MESSAGE */
00721  2                ERRORCOUNT = ERRORCOUNT + 1;
00722  3                CALL PRINTCHAR(HIGH(ERRCODE));
00723  3                CALL PRINTCHAR(LOW(ERRCODE));
00724  3                CALL PRINTCHAR(QUESTIONMARK);
00725  3                DO WHILE(POINTER:=POINTER - 1) >= 1;
00726  3                    CALL PRINTCHAR(' ');
00727  4                    END;
00728  3                CALL PRINTCHAR(UPARROW);
00729  3                CALL CRLF;
00730  3            END;
00731  2        CALL GET$NO$BLANK;
00732  2        ERRSET, COMPILING = FALSE;
00733  2        CALL SETFLAGS;
00734  2        RETURN;
00735  2      END ERROR;
00736  1
00737  1            /*
00738  1             ***************************************************************
00739  1             *                                                             *
00740  1             *     INITIALIZE$SCANNER SETS NEXTCHAR TO THE FIRST           *
00741  1             *     NON-BLANK CHARACTER ON THE INPUT FILE AND               *
00742  1             *     INITIALIZES THE OUTPUTLINE COUNTER AND POINTER          *
00743  1             *                                                             *
00744  1             *     INITIALIZE$SCANNER IS CALLED AT THE BEGINNING OF        *
00745  1             *     PASS ONE AND PASS TWO.                                  *
00746  1             *                                                             *
00747  1             ***************************************************************
00748  1            */
00749  1    INITIALIZE$SCANNER: PROCEDURE;
00750  2        DECLARE COUNT BYTE;
00751  2        CALL OPEN$SOURCEFILE;
00752  2        LINENO,LINEPTR = 0;
00753  2        CALL CLEAR$LINE$BUFF;
00754  2        SOURCEPTR = SOURCERECSIZE;
00755  2        SEPARATOR = COLIN;
00756  2        CALL GETNOBLANK;
00757  2        IF NEXTCHAR = '$' THEN
00758  2            DO;
00759  2                DO WHILE GETCHAR <> EOLCHAR;
00760  3                    IF(COUNT := (NEXTCHAR AND 5FH) - 'A') <= 4 THEN
00761  4                        DO CASE COUNT;
00762  4                            LISTPROD = TRUE;
00763  5                            LISTSOURCE = FALSE;
00764  5                            NOINTFILE = TRUE;
00765  5                            LOWERTOUPPER = FALSE;
00766  5                            DEBUGLN = TRUE;
00767  5                        END; /* OF CASE */
00768  4                END;
00769  3                CALL GETNOBLANK;
00770  3            END;
00771  2        RETURN;
00772  2    END INITIALIZE$SCANNER;
```

```
00773  1
00774  1
00775  1          /*
00776  1           ************************************************************
00777  1           *                                                          *
00778  1           *     THE SCANNER ACCEPTS INPUT CHARACTERS FROM THE        *
00779  1           *     SOURCE FILE RETURNING TOKENS TO THE PARSER.          *
00780  1           *     CONVERSION TO UPPERCASE IS PERFORMED WHEN SCAN-      *
00781  1           *     NING IDENTIFIERS UNLESS LOWERTOUPPER IS FALSE.       *
00782  1           *     BLANKS ARE IGNORED.  EACH TOKEN IS PLACED IN         *
00783  1           *     ACCUM.  ACCUM(0) IS THE LENGTH OF THE TOKEN.         *
00784  1           *     THE TOKEN IS HASHCODED BY SUMMING EACH ASCII         *
00785  1           *     CHARACTER MODULO HASHTBLSIZE AND THE RESULT IS       *
00786  1           *     RETURNED IN HASHCODE.  SUBTYPE AND FUNCOP ARE        *
00787  1           *     SET IF THE TOKEN IS A PREDEFINED FUNCTION.           *
00788  1           *     REM AND DATA STATEMENTS ARE HANDLED COMPLETELY       *
00789  1           *     BY THE SCANNER. IF THE RESERVED WORD REM OR          *
00790  1           *     REMARK IS DETECTED THE INPUT IS SCANNED UNTIL        *
00791  1           *     THE END OF THE CURRENT INPUT LINE IS LOCATED.        *
00792  1           *     THE NEXT TOKEN (A CARRIAGE RETURN) IS THEN           *
00793  1           *     SCANNED AND RTURNED. DATA STATEMENTS ARE SIMILAR     *
00794  1           *     EXCEPT THE DATA IS WRITTEN OUT USEING EMITDAT        *
00795  1           *                                                          *
00796  1           ************************************************************
00797  1          */
00798  1     SCANNER: PROCEDURE;
00799  2
00800  2          /*
00801  2           ************************************************************
00802  2           *                                                          *
00803  2           *     THE FOLLOWING UTILITY PROCEDURES ARE USED BY THE     *
00804  2           *     SCANNER.                                             *
00805  2           *                                                          *
00806  2           ************************************************************
00807  2          */
00808  2
00809  2          PUTINACCUM: PROCEDURE;
00810  3               IF NOT CONT THEN
00811  3                    DO;
00812  3                         ACCUM(ACCUM := ACCUM + 1) = NEXTCHAR;
00813  4                         HASHCODE = (HASHCODE + NEXTCHAR) AND HASHMASK;
00814  4                         IF ACCUM >= (IDENTSIZE - 1) THEN
00815  4                              CONT = TRUE;
00816  4                    END;
00817  3               RETURN;
00818  3          END PUTINACCUM;
00819  2
00820  2
00821  2          PUTANDGET: PROCEDURE;
00822  3               CALL PUTINACCUM;
00823  3               CALL GETNOBLANK;
00824  3               RETURN;
00825  3          END PUTANDGET;
00826  2
00827  2
00828  2          PUTANDCHAR: PROCEDURE;
00829  3               CALL PUTINACCUM;
00830  3               NEXTCHAR = GETCHAR;
00831  3               RETURN;
00832  3          END PUTANDCHAR;
00833  2
00834  2
00835  2          NUMERIC: PROCEDURE BYTE;
00836  3               RETURN(NEXTCHAR - '0') <= 9;
00837  3          END NUMERIC;
00838  2
00839  2          LOWERCASE: PROCEDURE BYTE;
00840  3               RETURN (NEXTCHAR >= 61H) AND (NEXTCHAR <= 7AH);
00841  3          END LOWERCASE;
00842  2
00843  2
00844  2          DECIMALPT: PROCEDURE BYTE;
00845  3               RETURN NEXTCHAR = '.';
00846  3          END DECIMALPT;
00847  2
00848  2
00849  2          CONV$TO$UPPER: PROCEDURE;
00850  3               IF LOWERCASE AND LOWERTOUPPER THEN
00851  3                    NEXTCHAR = NEXTCHAR AND 5FH;
00852  3               RETURN;
00853  3          END CONV$TO$UPPER;
00854  2
00855  2
00856  2          LETTER: PROCEDURE BYTE;
00857  3               CALL CONV$TO$UPPER;
00858  3               RETURN ((NEXTCHAR - 'A') <= 25) OR LOWERCASE;
00859  3          END LETTER;
00860  2
00861  2
00862  2          ALPHANUM: PROCEDURE BYTE;
00863  3               RETURN NUMERIC OR LETTER OR DECIMALPT;
00864  3          END ALPHANUM;
00865  2
00866  2
00867  2          SPOOLNUMERIC: PROCEDURE;
00868  3               DO WHILE NUMERIC;
00869  3                    CALL PUTANDCHAR;
00870  4                    END;
```

```
00871  3            RETURN;
00872  3        END SPOCLNUMERIC;
00873  2
00874  2
00875  2        SETUP$NEXT$CALL: PROCEDURE;
00876  3            IF NEXTCHAR = ' ' THEN
00877  3                CALL GETNOBLANK;
00878  3            CONT = FALSE;
00879  3            RETURN;
00880  3        END SETUP$NEXT$CALL;
00881  2
00882  2    EMITDAT: PROCEDURE(OBJCODE);
00883  3        /*
00884  3          WRITES DATA STATEMENTS DURING PASS2 AND
00885  3          COUNTS SIZE OF DATA AREA.
00886  3        */
00887  3        DECLARE OBJCODE BYTE;
00888  3        DATACT = DATACT + 1;
00889  3        IF PASS2 THEN
00890  3            CALL EMIT(OBJCODE);
00891  3        RETURN;
00892  3    END EMITDAT;
00893  2
00894  2        /*
00895  2         ************************************************************
00896  2         *                                                          *
00897  2         *       LOOKUP IS CALLED BY THE SCANNER WITH THE           *
00898  2         *        PRINTNAME OF THE CURRENT TOKEN IN                 *
00899  2         *       THE ACCUMULATOR.  LOOKUP DETERMINES IF THIS        *
00900  2         *       TOKEN IS A RESERVED WORD AND  SETS THE             *
00901  2         *       VALUE OF TOKEN.  IF THE TOKEN IS A PREDEFINED      *
00902  2         *       FUNCTION THEN THE SUBTYPE AND FUNCOP ARE ALSO      *
00903  2         *       SET.                                               *
00904  2         *       THE RESERVED WORD TABLE IS DIVIDED INTO 7          *
00905  2         *       TABLES FOR RESERVED WORDS OF LENGTH 1 TO 7.        *
00906  2         *       THE FOLLOWING VECTORS ARE ALSO USED:               *
00907  2         *           TK - TOKEN ASSOCIATED WITH RESERVED WORD       *
00908  2         *           OFFSET - INDEX INTO LNG VECTOR FOR A GIVEN     *
00909  2         *                    R/W LENGTH                            *
00910  2         *           COUNT - NUMBER OF R/W OF A GIVEN LENGTH        *
00911  2         *           TKOS - INDEX INTO TK FOR A GIVEN R/W LENGTH    *
00912  2         *           ST - SPECIAL DATA FOR PREDEFINED FUNCTIONS     *
00913  2         *                                                          *
00914  2         *       PREDEFINED FUNCTIONS HAVE TOKEN VALUES >64.        *
00915  2         *       THIS NUMBER BECOMES THE FUNCOP AND THE TOKEN       *
00916  2         *       IS FUNCT.  FUNCOP IS THE MACHINE CODE FOR THE      *
00917  2         *       PARTICULAR PREDEFINED FUNCTION.                    *
00918  2         *                                                          *
00919  2         ************************************************************
00920  2        */
00921  2
00922  2        LOOKUP: PROCEDURE BYTE;
00923  3
00924  3            DECLARE MAXRWLNG LIT '9';  /* MAX LENGTH OF A RESERVED WORD */
00925  3
00926  3            DECLARE LNG1 DATA(EOLCHAR,'<','(','+','*',')','-',',','=','/',
00927  3                              ';','>',':',POUNDSIGN,UPARROW),  /*  15  */
00928  3                    LNG2 DATA('IF','TO','GO','ON','OR','EQ','LT','GT',
00929  3                              'LE','GE','NE'),  /*  11  */
00930  3                    LNG3 DATA('FOR','LET','REM','DIM','DEF','NOT','AND',
00931  3                              'TAN','SIN','COS','SQR','TAB','LOG','LEN',
00932  3                              'FRE','ATN','ABS','EXP','INT','END','POS',
00933  3                              'RND','SGN','INP','ASC','VAL','XOR','SUB',
00934  3                               'OUT'),
00935  3                                        /*  29  */
00936  3                    LNG4 DATA('THEN','READ','GOTO','ELSE','NEXT',
00937  3                              'STOP','DATA','FILE','CHR$','MID$',
00938  3                              'STEP','STR$','COSH','SINH'),  /*  14  */
00939  3                    LNG5 DATA('PRINT','INPUT','GOSUB','CLOSE',
00940  3                                'LEFT$'),  /*  5  */
00941  3                    LNG6 DATA('RETURN','RIGHT$','REMARK'),  /* 3 */
00942  3                    LNG7 DATA('RESTORE'),  /*  1  */
00943  3                    LNG9 DATA('RANDOMIZE'),
00944  3                      TK DATA(0,TCR,LESST,LPARN,TPLUS,ASTRK,RPARN,TMINUS,
00945  3                              COMMA,EQUAL,SLASH,SCOLN,GTRT,TCOLIN,POUND,
00946  3                              EXPON,          /* LNG 1 */
00947  3                              TIF,TTO,TGO,TON,TOR,EQUAL,LESST,GTRT,TLEQ,
00948  3                              TGEQ,TNE,  /* LNG2 */
00949  3                              TFOR,TLET,TREM,TDIM,TDEF,TNOT,TAND,
00950  3                              72,69,70,73,74,78,84,76,71,65,75,
00951  3                              66,TEND,79,67,68,80,81,88,TXOR,TSUB,TOUT,
00952  3                                        /* LNG 3 */
00953  3                              TTHEN,TREAD,TGOTO,TELSE,TNEXT,TSTOP,TDATA,
00954  3                              TFILE,82,85,TSTEP,87,89,90,  /* LNG 4 */
00955  3                              TPRNT,TINPT,TGOSB,TCLOS,83,       /* LNG 5 */
00956  3                              TRETN,86,TREM,  /* LNG 6 */
00957  3                              TREST,TIRN),
00958  3                   OFFSET DATA(0,0,15,37,124,180,205,223,230,230),
00959  3                    COUNT DATA(0,15,11,29,14,5,3,1,0,1),
00960  3                     TKOS DATA(0,0,15,26,55,69,74,77,78,78),
00961  3                       ST DATA(1,1,0,1,1,1,1,1,1,1,0,0,1,0,1,
00962  3                               5,65,70,5,71,70,65,5,1,1);
00963  3
00964  3            DECLARE
00965  3                    PTR             ADDRESS,
00966  3                    FIELD BASED PTR BYTE,
00967  3                    I               BYTE;
```

```
00968  3
00969  3            COMPARE: PROCEDURE BYTE;
00970  4                 DECLARE I BYTE;
00971  4                 I = 0;
00972  4                 DO WHILE (FIELD(I) = ACCUM(I := I + 1)) AND I <= ACCUM;
00973  4                    END;
00974  4                 RETURN I > ACCUM;
00975  4            END COMPARE;
00976  3
00977  3            IF ACCUM > MAXRWLNG THEN
00978  3                 RETURN FALSE;
00979  3            PTR = OFFSET(ACCUM) + .LNG1;
00980  3            DO I = 1 TO COUNT(ACCUM);
00981  3                 IF COMPARE THEN
00982  4                      DO;
00983  4
00984  4                           IF((TOKEN := TK(TKOS(ACCUM) + I)) > 64) AND
00985  5                              (TOKEN <> TDATA) THEN
00986  5                                DO;
00987  5                                     SUBTYPE = ST(TOKEN - 65);
00988  6                                     FUNCOP = TOKEN;
00989  6                                     TOKEN = FUNCT;
00990  6                                END;
00991  5                           RETURN TRUE;
00992  5                      END;
00993  4                 PTR = PTR + ACCUM;
00994  4                 END;
00995  3            RETURN FALSE;
00996  3       END LOOKUP;
00997  2
00998  2
00999  2       DO FOREVER;  /* TO HANDLE REM, DAT AND CONTINUATION */
01000  2       ACCUM, HASHCODE, TOKEN, SUBTYPE = 0;
01001  3            /* FIRST CASE - IS THIS A STRING OR THE CONTINUATION
01002  3               OF A STRING   (ONLY STRINGS MAY BE CONTINUED)
01003  3            */
01004  3       IF(NEXTCHAR = STRINGDELIM) OR CONT THEN
01005  3            DO; /* FOUND STRING */
01006  3                 TOKEN = STRING;
01007  4                 CONT = FALSE;
01008  4                 DO FOREVER;  /* ALLOWS    IN STRING TO BE.  */
01009  4                      DO WHILE GETCHAR <> STRINGDELIM;
01010  5                           CALL PUTINACCUM;
01011  6                           IF CONT THEN RETURN;
01012  6                           END;
01013  5                      CALL GETNOBLANK;
01014  5                      IF NEXTCHAR <> STRINGDELIM THEN
01015  5                           RETURN;
01016  5                      CALL PUT$IN$ACCUM;
01017  5                 END; /* OF DO FOREVER */
01018  4            END; /* OF RECOGNIZING A STRING */
01019  3            /*
01020  3               NEXT CASE IS A NUMERIC WHICH MUST START WITH A
01021  3               NUMBER OR WITH A PERIOD
01022  3               ONLY FIRST IDENTSIZE CHARACTERS ARE RETAINED
01023  3            */
01024  3       ELSE IF NUMERIC OR DECIMALPT THEN
01025  3            DO; /* HAVE DIGIT */
01026  3                 TOKEN = FLOATPT;
01027  4                  DO WHILE NEXTCHAR = '0'; /* ELIM LEADING ZEROS */
01028  4                       NEXTCHAR = GETCHAR;
01029  5                       END;
01030  4                 CALL SPOOLNUMERIC;  /* GET ALL THE NUMBERS */
01031  4                 IF DECIMALPT THEN
01032  4                      DO;
01033  4                           CALL PUTANDCHAR;
01034  5                           CALL SPOOLNUMERIC;
01035  5                      END;
01036  4                 CALL CONV$TO$UPPER;
01037  4                 IF NEXTCHAR = 'E' THEN
01038  4                      DO;  /* A FLOATING POINT NUMBER */
01039  4                           CALL PUTANDGET;
01040  5                           IF (NEXTCHAR = '+') OR (NEXTCHAR='-') THEN
01041  5                                CALL PUTANDGET;
01042  5                           IF NOT NUMERIC THEN
01043  5                                CALL ERROR('IF');
01044  5                           CALL SPOOL$NUMERIC;
01045  5                      END;
01046  4                  IF ACCUM = 0 THEN
01047  4                       HASHCODE, ACCUM(ACCUM := 1) = '0';
01048  4                 CALL SETUP$NEXT$CALL;
01049  4                 RETURN;
01050  4            END; /* OF RECOGNIZING NUMERIC CONSTANT */
01051  3            /*
01052  3               NEXT CASE IS IDENTIFIER. MAY BE RESERVED WORD
01053  3               IN WHICH CASE MAY BE REM, OR DATA. THESE STATEMENTS
01054  3               ARE HANDLED BY THE SCANNER VICE THE PARSER AND THEN
01055  3               ANOTHER LOOP THROUGH THE SCANNER IS MADE.
01056  3               ONLY IDENTSIZE-1 CHARACTERS ARE RETAINED
01057  3            */
01058  3       ELSE IF LETTER THEN
01059  3            DO; /* HAVE A LETTER */
01060  3                 DO WHILE ALPHANUM;
01061  3                      CALL PUTANDCHAR;
01062  4                      END;
01063  5                 IF NEXTCHAR = '$' THEN
01064  4                      DO;
01065  4                           SUBTYPE = STRING;
01066  4                           CALL PUTANDCHAR;
01067  5
```

```
01068  5                    END;
01069  4                ELSE
01070  4                    SUBTYPE = FLOATPT;
01071  4                IF NOT LOOKUP THEN
01072  4                    DO;
01073  4                        IF ACCUM(1) = 'F' AND ACCUM(2) = 'N'
01074  5                            AND ACCUM <> 1 THEN
01075  5                            TOKEN = UDFUNCT;
01076  5                        ELSE
01077  5                            TOKEN = IDENTIFIER;
01078  5                        CALL SETUP$NEXT$CALL;
01079  5                        RETURN;
01080  5                    END;
01081  4                ELSE /* IS A RW */
01082  4                    IF TOKEN = TREM THEN
01083  4                        DO WHILE NEXTCHAR <> EOLCHAR;
01084  4                            NEXTCHAR = GETCHAR;
01085  5                            CALL CHECK$CONTINUATION;
01086  5                            END;
01087  4                    ELSE
01088  4                        IF TOKEN = TDATA THEN
01089  4                            DO;
01090  4                                DECLARE DAT LIT '51';
01091  5                                CALL EMITDAT(DAT);
01092  5                                CALL EMITDAT(NEXTCHAR);
01093  5                                DO WHILE GETCHAR <> EOLCHAR;
01094  5                                    CALL CHECK$CONTINUATION;
01095  6                                    CALL EMITDAT(NEXTCHAR);
01096  6                                    END;
01097  5                                CALL EMITDAT(',');
01098  5                                CALL EMITDAT(0);
01099  5                                DATACT = DATACT - 1;
01100  5                            END;
01101  4                        ELSE
01102  4                            DO;
01103  4                                CALL SETUP$NEXT$CALL;
01104  5                                RETURN;
01105  5                            END;
01106  4            END; /* OF RECOGNIZING RW OR IDENT */
01107  3           /*
01108  3               LAST CASE IS A SPECIAL CHARACTER - IT MAY BE
01109  3               THE CONTINUATION CHARACTER IN WHICH CASE JUST
01110  3               GO TO NEXT LINE AND SCAN SOMEMORE.
01111  3           */
01112  3        ELSE
01113  3            DO; /* SPECIAL CHARACTER */
01114  3                IF NEXTCHAR = CONTCHAR THEN
01115  4                    CALL CHECK$CONTINUATION;
01116  4                ELSE
01117  4                    DO;
01118  4                        CALL PUTANDGET;
01119  5                        IF NOT LOOKUP THEN
01120  5                            CALL ERROR('IC');
01121  5                        RETURN;
01122  5                    END;
01123  4            END; /* OF RECOGNIZING SPECIAL CHAR */
01124  3        END;  /* OF DO FOREVER */
01125  2
01126  2    END SCANNER;
01127  1
01128  1        /*
01129  1         ************************************************************
01130  1         *                                                          *
01131  1         *     SYMBOL TABLE PROCEDURES                              *
01132  1         *                                                          *
01133  1         *     THE SYMBOL TABLE IS BUILT FROM .MEMORY TOWARD        *
01134  1         *     THE LARGEST USABLE ADDRESS WHICH IS STORED IN MAX.   *
01135  1         *     INFORMATION REQUIRED DURING FOR STATEMENT CODE       *
01136  1         *     GENERATION IS MAINTAINED STARTING AT MAX AND         *
01137  1         *     WORKING DOWN TOWARD THE TOP OF THE SYMBOL TABLE      *
01138  1         *     THE FOLLOWING ARE MAJOR GLOBAL VARIABLES USED        *
01139  1         *     BY THE SYMBOL TABLE AND THEIR MEANING:               *
01140  1         *          SBTBLTOP - CURRENT POSITION OF FOR/NEXT         *
01141  1         *                     STACK.                               *
01142  1         *          SBTBL - CURRENT  TOP  OF SYMBOL TABLE           *
01143  1         *          BASE - ADDRESS OF BEGINNING OF ENTRY. THIS      *
01144  1         *                 MUST BE SET BEFORE AN ENTRY MAY BE       *
01145  1         *                 ACCESSED.                                *
01146  1         *          PRINTNAME - ADDRESS OF PRINTNAME OF AN ENTRY    *
01147  1         *                      TO BE USED IN REFERENCE TO THE      *
01148  1         *                      SYMBOL TABLE.                       *
01149  1         *          SYMHASH - HASH OF TOKEN REFERENCE BY            *
01150  1         *                    PRINTNAME                             *
01151  1         *                                                          *
01152  1         *     THE FOLLOWING IS THE STRUCTURE OF A SYMBOL           *
01153  1         *     TABLE ENTRY:                                         *
01154  1         *           LENGTH OF PRINTNAME   - 1 BYTE                 *
01155  1         *           COLLISION FIELD   -   2 BYTES                  *
01156  1         *           PRINTNAME     -  VARIABLE LENGTH               *
01157  1         *           TYPE      -   1 BYTE                           *
01158  1         *               LEFTMOST BIT OF THIS BYTE IS A FLAG        *
01159  1         *               TO INDICATE IF THE ADDRESS HAS BEEN        *
01160  1         *               SET.                                       *
01161  1         *           LOCATION - 2 BYTES                             *
01162  1         *           SUBTYPE - 1 BYTES                              *
01163  1         *                                                          *
01164  1         *       THE FOLLOWING GLOBAL ROUTINES ARE PROVIDED         *
```

```
01165  1              *       FOR SYMBOL TABLE MANIPULATION:                  *
01166  1              *          LOOKUP       ENTER      GETLEN      GETYPE    *
01167  1              *          SETYPE       GETRES     GETADDR     SETADDR   *
01168  1              *          SETSUBTYPE   GETSUBTYPE UNLINK      RELINK    *
01169  1              *                                                        *
01170  1              **********************************************************
01171  1             */
01172  1
01173  1
01174  1     INITIALIZE$SYMTBL: PROCEDURE;
01175  2         /* FILL HASHTABLE WITH 0'S */
01176  2         IF PASS1 THEN
01177  2              DO;
01178  2                   CALL FILL(.HASHTABLE,0,SHL(HASHTBLSIZE,2));
01179  3                   SBTBL = .MEMORY;
01180  3              END;
01181  2         /* INITIALIZE POINTER TO TOP OF SYMBOL TABLE */
01182  2         SBTBLTOP, NEXTSTMTPTR = MAX - 2;
01183  2         NEXTBYTE(1) =0;
01184  2         RETURN;
01185  2     END INITIALIZE$SYMTBL;
01186  1
01187  1     SETADDRPTR: PROCEDURE(OFFSET);   /* SET PTR FOR ADDR REFERENCE */
01188  2         DECLARE
01189  2                   OFFSET BYTE;
01190  2         APTRADDR = BASE + PTR + OFFSET; /* POSITION FOR ADDR REFERENCE */
01191  2         RETURN;
01192  2     END SETADDRPTR;
01193  1
01194  1
01195  1     GETHASH: PROCEDURE BYTE;
01196  2         DECLARE HASH BYTE,
01197  2                 I    BYTE;
01198  2         HASH = 0;
01199  2         APTRADDR = BASE + 2;
01200  2         DO I = 1 TO PTR;
01201  2              HASH = (HASH + BYTEPTR(I)) AND HASHMASK;
01202  3              END;
01203  2         RETURN HASH;
01204  2     END GETHASH;
01205  1
01206  1
01207  1     NEXTENTRY: PROCEDURE;
01208  2         BASE = BASE + PTR + 7;
01209  2         RETURN;
01210  2     END NEXTENTRY;
01211  1
01212  1
01213  1     SETLINK: PROCEDURE;
01214  2         APTRADDR = BASE + 1;
01215  2         RETURN;
01216  2     END SETLINK;
01217  1
01218  1
01219  1     HASHTBL$OF$SYMHASH: PROCEDURE ADDRESS;
01220  2         RETURN HASHTABLE(SYMHASH);
01221  2     END HASHTBL$OF$SYMHASH;
01222  1
01223  1     LIMITS: PROCEDURE(COUNT);
01224  2              /*
01225  2               CHECK TO SEE IF ADDITIONAL SBTBL WILL OVERFLOW LIMITS OF
01226  2               MEMORY. IF SO THEN PUNT ELSE RETURN
01227  2              */
01228  2
01229  2               DECLARE COUNT  BYTE;      /*SIZE BEING ADDED IS COUNT  */
01230  2               IF SBTBLTOP <= (SBTBL + COUNT) THEN
01231  2                    DO;
01232  2                         PASS2 = TRUE;  /* TO PRINT ERROR MSG */
01233  3                         CALL ERROR('TO');
01234  3                          CALL MON3;
01235  3                    END;
01236  2         RETURN;
01237  2     END LIMITS;
01238  1
01239  1
01240  1     SETADDR: PROCEDURE(LOC);
01241  2              /*SET THE ADDRESS FIELD AND RESOLVED BIT*/
01242  2         DECLARE    LOC         ADDRESS;
01243  2         CALL SETADDRPTR (4);
01244  2         ADDRPTR=LOC;
01245  2         APTRADDR = APTRADDR - 1;
01246  2         BYTEPTR=BYTEPTR OR 80H;
01247  2         RETURN;
01248  2     END SETADDR;
01249  1
01250  1
01251  1     LOCKUP: PROCEDURE BYTE;
01252  2               /*
01253  2                CHECK TO SEE IF P/N LOCATED AT ADDR IN PRINTNAME IS IN SBTBL
01254  2                        RETURN TRUE IF IN SBTBL
01255  2                        RETURN FALSE IF NOT IN SBTBL.
01256  2                        BASE=ADDRESS IF IN SBTBL
01257  2               */
01258  2
01259  2           DECLARE
01260  2                   LEN                    BYTE,
01261  2                   N BASED PRINTNAME BYTE; /* N IS LENGTH OF P/N */
01262  2          BASE = HASHTBL$OF$SYMHASH;
```

```
01263  2          DO WHILE BASE <> 0;
01264  2               IF(LEN := PTR) = N THEN
01265  3               DO WHILE (PTR(LEN + 2) = N(LEN));
01266  3                    IF (LEN := LEN - 1) = 0 THEN
01267  4                         RETURN TRUE;
01268  4               END;
01269  3               CALL SETLINK;
01270  3               BASE = ADDRPTR;
01271  3          END;
01272  2          RETURN FALSE;
01273  2     END LOOKUP;
01274  1
01275  1
01276  1     ENTER: PROCEDURE;
01277  2               /*
01278  2                  ENTER TOKEN REFERENCE BY PRINTNAME AND SYMHASH
01279  2                  INTO NEXT AVAILABLE LOCATION IN THE SYMBOL TABLE.
01280  2                  SET BASE TO BEGINNING OF THIS ENTRY AND INCREMENT
01281  2                  SBTBL.  ALSO CHECK FOR SYMBOL TABLE FULL.
01282  2               */
01283  2          DECLARE
01284  2                  I                    BYTE,
01285  2                  N BASED PRINTNAME BYTE;
01286  2          CALL LIMITS(I:=N+7);
01287  2          BASE = SBTBL; /* BASE FOR NEW ENTRY */
01288  2          CALL MOVE(PRINTNAME + 1,SBTBL + 3,(PTR := N));
01289  2          CALL SETADDRPTR(3);/* SET RESOLVE BIT TO 0 */
01290  2          BYTEPTR = 0;
01291  2          CALL SETLINK;
01292  2          ADDRPTR = HASHTBL$OF$SYMHASH;
01293  2          HASHTABLE(SYMHASH) = BASE;
01294  2          SBTBL = SBTBL + I;
01295  2          RETURN;
01296  2     END ENTER;
01297  1
01298  1
01299  1     GETLEN: PROCEDURE       BYTE;      /*RETURN LENGTH OF THE P/N  */
01300  2          RETURN PTR;
01301  2     END GETLEN;
01302  1
01303  1
01304  1     GETYPE: PROCEDURE       BYTE;           /*RETURNS TYPE OF VARIABLE  */
01305  2        CALL SETADDRPTR (3);
01306  2        RETURN (BYTEPTR AND 7FH);
01307  2     END GETYPE;
01308  1
01309  1
01310  1     SETYPE: PROCEDURE (TYPE);         /*SET TYPEFIELD = TYPE  */
01311  2          DECLARE    TYPE      BYTE;
01312  2          CALL SETADDRPTR (3);
01313  2          BYTEPTR = BYTEPTR OR TYPE;
01314  2               /*THIS SETS THE TYPE AND PRESERVES RESOLVED BIT */
01315  2          RETURN;
01316  2     END SETYPE;
01317  1
01318  1
01319  1     GETRES: PROCEDURE  BYTE;
01320  2               /*
01321  2                 RETURN TRUE IF RESOLVED BIT = 1,
01322  2                 RETURN FALSE IF RESOLVED BIT = 0
01323  2               */
01324  2          CALL SETADDRPTR(3);
01325  2          RETURN ROL(BYTEPTR,1);
01326  2     END GETRES;
01327  1
01328  1
01329  1     GETADDR: PROCEDURE ADDRESS;
01330  2               /*RETURN THE ADDRESS OF THE P/N LOCATION */
01331  2          CALL SETADDRPTR(4);
01332  2          RETURN ADDRPTR;
01333  2     END GETADDR;
01334  1
01335  1
01336  1     SETSUBTYPE: PROCEDURE(STYPE);       /*INSERT THE SUBTYPE IN SBTBL */
01337  2          DECLARE    STYPE     BYTE;
01338  2          CALL SETADDRPTR (6);
01339  2          BYTEPTR=STYPE;
01340  2          RETURN;
01341  2     END SETSUBTYPE;
01342  1
01343  1
01344  1     GETSUBTYPE: PROCEDURE BYTE;          /*RETURN THE SUB TYPE */
01345  2          CALL SETADDRPTR (6);
01346  2          RETURN BYTEPTR;
01347  2     END GETSUBTYPE;
01348  1
01349  1
01350  1     UNLINK: PROCEDURE;
01351  2          DECLARE NEXTA   ADDRESS,
01352  2                  NUMPARM BYTE,
01353  2                  I       BYTE,
01354  2                  ENTRYPT BASED NEXTA ADDRESS;
01355  2          NUMPARM = GETYPE;
01356  2          DO I = 1 TO NUMPARM;
01357  2               CALL NEXTENTRY;
01358  3               NEXTA = SHL(GETHASH,1) + .HASHTABLE; /* ITS ON THIS CHAIN */
01359  3               DO WHILE ENTRYPT <> BASE;
01360  3                    NEXTA = ENTRYPT + 1;
```

```
01361  4                END;
01362  3             CALL SETLINK;
01363  3             ENTRYPT = ADDRPTR;
01364  3         END;
01365  2         RETURN;
01366  2     END UNLINK;
01367  1
01368  1
01369  1     RELINK: PROCEDURE;
01370  2         DECLARE
01371  2                 TEMPA           ADDRESS,
01372  2                 I               BYTE,
01373  2                 NUMPARM         BYTE,
01374  2                 LOC BASED TEMPA ADDRESS;
01375  2         NUMPARM = GETYPE;
01376  2         DO I = 1 TO NUMPARM;
01377  2             CALL NEXTENTRY;
01378  3             TEMPA = BASE + I;
01379  3             LOC = HASHTABLE(GETHASH);
01380  3             HASHTABLE(GETHASH) = BASE;
01381  3             END;
01382  2         RETURN;
01383  2     END RELINK;
01384  1
01385  1
01386  1             /*
01387  1              ***************************************************
01388  1              *                                                 *
01389  1              *   ****   PARSER AND CODE GENERATION  SECTION  **** *
01390  1              *                                                 *
01391  1              ***************************************************
01392  1             */
01393  1     DO;
01394  1
01395  1              /*
01396  1                MNUMONICS FOR NBASIC MACHINE
01397  1              */
01398  1     DECLARE
01399  2             FAD LIT '0', DUP LIT '18', WST LIT '36',
01400  2             FMI LIT '1', XCH LIT '19', RDF LIT '37',
01401  2             FMU LIT '2', STD LIT '20', RDB LIT '38',
01402  2             FDI LIT '3', SLT LIT '21', ECR LIT '39',
01403  2             EXP LIT '4', SGT LIT '22', WRB LIT '40',
01404  2             LSS LIT '5', SEQ LIT '23', RDN LIT '41',
01405  2             GTR LIT '6', SNE LIT '24', RDS LIT '42',
01406  2             EQU LIT '7', SGE LIT '25', WRN LIT '43',
01407  2             NEQ LIT '8', SLE LIT '26', WRS LIT '44',
01408  2             GEQ LIT '9', STS LIT '27', OPN LIT '45',
01409  2             LEQ LIT '10', ILS LIT '28', CON LIT '46',
01410  2            NOTO LIT '11', CAT LIT '29', RST LIT '47',
01411  2            ANDO LIT '12', PRO LIT '30', NEG LIT '48',
01412  2             BOR LIT '13', RTN LIT '31', RES LIT '49',
01413  2             LOD LIT '14', ROW LIT '32', NOP LIT '50',
01414  2             STO LIT '15', SUBO LIT '33', DAT LIT '51',
01415  2             XIT LIT '16', RDV LIT '34', DBF LIT '52',
01416  2             DEL LIT '17', WRV LIT '35', NSP LIT '53',
01417  2             BRS LIT '54', BRC LIT '55', BFC LIT '56',
01418  2             BFN LIT '57', CV3 LIT '58', RCN LIT '59',
01419  2             DRS LIT '60', DRF LIT '61', EDR LIT '62',
01420  2             EDW LIT '63', CLS LIT '64', RON LIT '91',
01421  2             CKO LIT '92', EXR LIT '93', DEF LIT '94',
01422  2             BOL LIT '95', ADJ LIT '96', POT LIT '40',
01423  2             IRN LIT '77';
01424  2     DECLARE
01425  2             STATE STATESIZE,
01426  2              /*
01427  2                 THE FOLLOWING VECTORS ARE USED AS PARSE STACKS
01428  2                 SYNTHESIZE AND THE PARSER ACCESS THESE ARRAYS
01429  2              */
01430  2             STATESTACK(PSTACKSIZE) STATESIZE,
01431  2             HASH(PSTACKSIZE) BYTE,
01432  2             SYMLOC(PSTACKSIZE) ADDRESS,
01433  2             SRLOC(PSTACKSIZE) ADDRESS,
01434  2             VAR(PSTACKSIZE) BYTE,
01435  2             TYPE(PSTACKSIZE) BYTE,
01436  2             STYPE(PSTACKSIZE) BYTE,
01437  2             VARC(VARCSIZE) BYTE,
01438  2             ONSTACK(MAXONCOUNT) BYTE,
01439  2             VARINDEX BYTE,  /* INDEX INTO VAR */
01440  2             SP          BYTE,
01441  2             MP          BYTE,
01442  2             MPP1        BYTE,
01443  2             NOLOOK      BYTE,
01444  2             IFLABLNG BYTE INITIAL(2),
01445  2              /*
01446  2                 THE FOLLOWING VARABLES ARE USED TO GENERATE
01447  2                 COMPILER LABELS.
01448  2              */
01449  2             IFLAB2 BYTE INITIAL(23),
01450  2             IFLABLE BYTE;
01451  2
01452  2     EMITCON: PROCEDURE(CHAR);
01453  3          /*
01454  3            WRITES NUMERIC CONSTANTS DURING PASS1
01455  3          */
01456  3         DECLARE CHAR BYTE;
01457  3         IF PASS1 THEN
01458  3             CALL EMIT(CHAR);
```

```
01459  3        RETURN;
01460  3    END EMITCON;
01461  2
01462  2    INITIALIZE$SYNTHESIZE: PROCEDURE;
01463  3        DECLARE CONZERO DATA(01H,30H);
01464  3        DECLARE CONONE DATA(01H,31H);
01465  3        CODESIZE,DATACT,ONSTACK,IFLABLE = 0;
01466  3        FDACT = 1;
01467  3        PRTCT = 0FFFFH;
01468  3        CALL SET$FLAGS;
01469  3        IF PASS1 THEN
01470  3             DO;
01471  3                  CALL SETUP$INT$FILE;
01472  4                  PRINTNAME = .CONONE;
01473  4                  SYMHASH = 31H;
01474  4                  CALL ENTER;
01475  4                  CALL EMITCON(31H);
01476  4                  CALL EMITCON('$');
01477  4                  CALL SETADDR(0);  /* CONSTANT 1 IS AT FOA PCS 0 */
01478  4                  CALL SETYPE(4); /* TYPE CONST */
01479  4                  PRINTNAME = .CONZERO;
01480  4                  SYMHASH = 30H;
01481  4                  CALL ENTER;
01482  4                  CALL EMITCON(30H);
01483  4                  CALL EMITCON('$');
01484  4                  CALL SETADDR(1);
01485  4                  CALL SETYPE(4);
01486  4             END;
01487  3        RETURN;
01488  3    END INITIALIZE$SYNTHESIZE;
01489  2
01490  2
01491  2    SYNTHESIZE: PROCEDURE(PRODUCTION);
01492  3        DECLARE
01493  3                PRODUCTION BYTE;
01494  3
01495  3
01496  3         DECLARE
01497  3                 /*
01498  3                   THESE LITERALS DEFINE DIFFERENT  TYPES  WHICH
01499  3                   MAY BE PLACED IN THE TYPE FIELD OF THE SYMBOL
01500  3                   TABLE BY ROUTINES IN SYNTHESIZE
01501  3                 */
01502  3                  SIMVAR LIT '00H',
01503  3                  SUBVAR LIT '02',
01504  3                  CONST LIT '04',
01505  3                  LABLE LIT '08',
01506  3                  UNFUNC LIT '0AH';
01507  3
01508  3         DECLARE
01509  3                   /*
01510  3                     THE FOLLOWING VARIABLES ARE USED TO HOLD THE
01511  3                     CONTENTS OF THE PARSE STACKS DURING EXECUTION
01512  3                     OF SYNTHESIZE. THE PROCEDURE COPY IS CALLED
01513  3                     TO UPDATE EACH OF THESE VARIABLES ON EACH CALL
01514  3                     TO SYNTHESIZE.  THIS REDUCES THE NUMBER OF
01515  3                     SUBSCRIPT REFERENCES REQUIRED
01516  3                   */
01517  3                  (TYPESP,TYPEMP,TYPEMP1) BYTE,
01518  3                  (STYPESP,STYPEMP,STYPEMP1) BYTE,
01519  3                  (HASHSP,HASHMP,HASHMP1) BYTE,
01520  3                  (SYMLOCSP,SYMLOCMP, SYMLOCMP1) ADDRESS,
01521  3                  (SRLOCSP,SRLOCMP) ADDRESS;
01522  3
01523  3            /*
01524  3             *************************************************************
01525  3             *                                                           *
01526  3             *    THE FOLLOWING PROCEDURES ARE USED BY SYTHESIZE         *
01527  3             *    TO GENERATE CODE REQUIRED BY THE PRODUCTIONS           *
01528  3             *                                                           *
01529  3             *     THE FIRST GROUP OF PROCEDURES CONSISTING OF           *
01530  3             *     COPY AND THE SET-------- PROCEDURES ARE USED          *
01531  3             *     TO PREVENT THE LARGE AMOUNT OF SUBSCRIPTING           *
01532  3             *     THAT WOULD BE REQUIRED TO ACCESS THE PARSE            *
01533  3             *     STACKS DURING CODE GENERATION.                        *
01534  3             *                                                           *
01535  3             *     THE REMAINING PROCEDURES DIRECTLY SUPPORT CODE        *
01536  3             *     GENERATION AND ARE ARRANGED IN LOGICAL GROUPS         *
01537  3             *     SUCH AS THOSE WHICH ASSIST IN ACCESSING THE           *
01538  3             *     SYMBOL TABLE OR THOSE USED TO GENERATE INTERNAL       *
01539  3             *     COMPILER LABLES.                                      *
01540  3             *                                                           *
01541  3             *************************************************************
01542  3            */
01543  3         COPY: PROCEDURE;
01544  4              TYPESP = TYPE(SP);
01545  4              TYPEMP1 = TYPE(MPP1);
01546  4              TYPEMP = TYPE(MP);
01547  4              STYPESP = STYPE(SP);
01548  4              STYPEMP1 = STYPE(MPP1);
01549  4              STYPEMP = STYPE(MP);
01550  4              SYMLOCSP = SYMLOC(SP);
01551  4              SYMLOCMP1 = SYMLOC(MPP1);
01552  4              SYMLOCMP = SYMLOC(MP);
01553  4              HASHMP = HASH(MP);
01554  4              HASHMP1 = HASH(MPP1);
01555  4              HASHSP = HASH(SP);
01556  4              SRLOCSP = SRLOC(SP);
01557  4              SRLOCMP = SRLOC(MP);
```

```
01558  4            RETURN;
01559  4         END COPY;
01560  3
01561  3
01562  3         SETSYMLOCSP: PROCEDURE(A);
01563  4              DECLARE A ADDRESS;
01564  4              SYMLOC(SP) = A;
01565  4              RETURN;
01566  4         END SETSYMLOCSP;
01567  3
01568  3
01569  3         SETSYMLOCMP: PROCEDURE(A);
01570  4              DECLARE A ADDRESS;
01571  4              SYMLOC(MP) = A;
01572  4              RETURN;
01573  4         END SETSYMLOCMP;
01574  3
01575  3
01576  3         SETTYPESP: PROCEDURE(B);
01577  4              DECLARE B BYTE;
01578  4              TYPE(SP) = B;
01579  4              RETURN;
01580  4         END SETTYPESP;
01581  3
01582  3
01583  3         SETSTYPESP: PROCEDURE(B);
01584  4              DECLARE B BYTE;
01585  4              STYPE(SP) = B;
01586  4              RETURN;
01587  4         END SETSTYPESP;
01588  3
01589  3
01590  3         SETSTYPEMP: PROCEDURE(B);
01591  4              DECLARE B BYTE;
01592  4              STYPE(MP) = B;
01593  4              RETURN;
01594  4         END SETSTYPEMP;
01595  3
01596  3
01597  3         SETTYPEMP: PROCEDURE(B);
01598  4              DECLARE B BYTE;
01599  4              TYPE(MP) = B;
01600  4              RETURN;
01601  4         END SETTYPEMP;
01602  3
01603  3
01604  3         SETHASHMP: PROCEDURE(B);
01605  4              DECLARE B BYTE;
01606  4              HASH(MP) = B;
01607  4              RETURN;
01608  4         END SETHASHMP;
01609  3
01610  3
01611  3         SETHASHSP: PROCEDURE(B);
01612  4              DECLARE B BYTE;
01613  4              HASH(SP) = B;
01614  4              RETURN;
01615  4         END SETHASHSP;
01616  3
01617  3
01618  3         SETSRLOCSP: PROCEDURE(A);
01619  4              DECLARE A ADDRESS;
01620  4              SRLOC(SP) = A;
01621  4              RETURN;
01622  4         END SETSRLOCSP;
01623  3
01624  3    GENERATE: PROCEDURE(OBJCODE);
01625  4           /*
01626  4              WRITES GENERATED CODE AND COUNTS SIZE
01627  4              OF CODE AREA.
01628  4           */
01629  4         DECLARE OBJCODE BYTE;
01630  4         CODESIZE = CODESIZE + 1;
01631  4         IF NOT PASS1 THEN
01632  4              CALL EMIT(OBJCODE);
01633  4         RETURN;
01634  4    END GENERATE;
01635  3
01636  3         CALC$VARC: PROCEDURE(B) ADDRESS;
01637  4              DECLARE B BYTE;
01638  4              RETURN VAR(B) + .VARC;
01639  4         END CALC$VARC;
01640  3
01641  3
01642  3         SETLOOKUP: PROCEDURE(A);
01643  4              DECLARE A BYTE;
01644  4              PRINTNAME = CALC$VARC(A);
01645  4              SYMHASH = HASH(A);
01646  4              RETURN;
01647  4         END SETLOOKUP;
01648  3
01649  3
01650  3         LOOKUP$ONLY: PROCEDURE(A) BYTE;
01651  4              DECLARE A BYTE;
01652  4              CALL SETLOOKUP(A);
01653  4              IF LOOKUP THEN
01654  4                   RETURN TRUE;
01655  4              RETURN FALSE;
01656  4         END LOOKUP$ONLY;
```

```
01657  3
01658  3
01659  3       NORMAL$LOOKUP: PROCEDURE(A) BYTE;
C1660  4            DECLARE A BYTE;
01661  4            IF LOOKUP$ONLY(A) THEN
01662  4                 RETURN TRUE;
01663  4            CALL ENTER;
01664  4            RETURN FALSE;
01665  4        END NORMAL$LOOKUP;
01666  3
01667  3
01668  3       COUNTPRT: PROCEDURE ADDRESS;
01669  4       /* COUNTS THE SIZE OF THE PRT */
C1670  4            RETURN (PRTCT := PRTCT + 1);
01671  4       END COUNTPRT;
01672  3
01673  3
01674  3        GENTWO: PROCEDURE(A);
01675  4          /* WRITES TWO BYTES OF OBJECT CODE ON DISK FOR LITERALS */
01676  4            DECLARE A ADDRESS;
C1677  4            CALL GENERATE(HIGH(A));
01678  4            CALL GENERATE(LOW(A));
01679  4            RETURN;
C1680  4        END GENTWO;
01681  3
01682  3
01683  3        LITERAL: PROCEDURE(A);
C1684  4            DECLARE A ADDRESS;
01685  4            CALL GENTWO(A OR 8000H);
01686  4            RETURN;
01687  4        END LITERAL;
01688  3
01689  3
C1690  3        LITLOAD: PROCEDURE(A);
01691  4            DECLARE A ADDRESS;
01692  4            CALL GENTWO(A OR 0C000H);
01693  4            RETURN;
01694  4        END LITLOAD;
01695  3
01696  3
01697  3       LINE$NUMBER: PROCEDURE;
01698  4            IF DEBUGLN THEN
01699  4                 DO;
01700  4                      CALL LITERAL(LINENO);
01701  5                      CALL GENERATE(BOL);
01702  5                 END;
01703  4            RETURN;
01704  4       END LINE$NUMBER;
01705  3
01706  3
01707  3       SETIFNAME: PROCEDURE;
01708  4            PRINTNAME = .IFLABLNG;
01709  4            SYMHASH = IFLABLE AND HASHMASK;
C1710  4            RETURN;
01711  4       END SETIFNAME;
01712  3
01713  3
01714  3       ENTER$COMPILER$LABEL: PROCEDURE(B);
01715  4            DECLARE B BYTE;
01716  4            IF PASS1 THEN
01717  4                 DO;
C1718  4                      CALL SETIFNAME;
01719  5                      CALL ENTER;
01720  5                      CALL SETADDR(CODESIZE + B);
01721  5                 END;
01722  4            RETURN;
01723  4       END ENTER$COMPILER$LABEL;
01724  3
01725  3
01726  3       SET$COMPILER$LABEL: PROCEDURE;
01727  4            DECLARE X BYTE;
C1728  4            IFLABLE = IFLABLE + 1;
C1729  4            CALL SETIFNAME;
C1730  4            X = LOOKUP;
01731  4            RETURN;
01732  4       END SET$COMPILER$LABEL;
01733  3
01734  3
01735  3       COMPILER$LABEL: PROCEDURE;
01736  4            CALL SET$COMPILER$LABEL;
01737  4            CALL GEN$TWO(GETADDR);
C1738  4            RETURN;
C1739  4       END COMPILER$LABEL;
C1740  3
01741  3
01742  3       CHKTYP1: PROCEDURE BYTE; /* CHECK MP,SP BOTH FLOATING PT */
01743  4            IF((STYPEMP <> FLOATPT) OR (STYPESP <> FLOATPT)) THEN
01744  4                 DO;
01745  4                      CALL ERROR('MF');
01746  5                      RETURN FALSE;
01747  5                 END;
01748  4            RETURN TRUE;
01749  4       END CHKTYP1;
01750  3
01751  3
01752  3       CHKTYP2: PROCEDURE BYTE; /* CHECK MP,SP BOTH SAME TYPE */
01753  4            IF STYPESP <> STYPEMP THEN
01754  4                 DO;
```

```
01755  4                         CALL ERROR('MM');
01756  5                         RETURN FALSE;
01757  5                    END;
01758  4                    RETURN TRUE;
01759  4          END CHKTYP2;
01760  3
01761  3
01762  3          CHKTYP3: PROCEDURE BYTE;
01763  4               CALL SETSTYPEMP(STYPESP);
01764  4               IF STYPESP = FLOATPT THEN
01765  4                    RETURN TRUE;
01766  4               CALL ERROR('MF');
01767  4               RETURN FALSE;
01768  4          END CHKTYP3;
01769  3
01770  3          CHKTYP4: PROCEDURE;
01771  4               IF STYPEMP1 = STRING THEN
01772  4                    CALL ERROR('MF');
01773  4               CALL SETTYPEMP(TYPEMP := TYPEMP + 1);
01774  4               CALL GENERATE(RON);
01775  4               RETURN;
01776  4          END CHKTYP4;
01777  3
01778  3
01779  3          SUBCALC: PROCEDURE;
01780  4               CALL SETSUBTYPE(TYPESP);
01781  4               CALL GENERATE(ROW);
01782  4               CALL GENERATE(TYPESP);
01783  4               CALL GENERATE(STD);
01784  4               RETURN;
01785  4          END SUBCALC;
01786  3
01787  3
01788  3          GEN$STORE: PROCEDURE;
01789  4               IF STYPEMP1 = FLOATPT THEN
01790  4                    CALL GENERATE(STD);
01791  4               ELSE
01792  4                    CALL GENERATE(STS);
01793  4               RETURN;
01794  4          END GEN$STORE;
01795  3
01796  3
01797  3          SETUP$INPUT: PROCEDURE;
01798  4               CALL GENERATE(DBF);
01799  4               INPUTSTMT = TRUE;
01800  4               CALL GENERATE(RCN);
01801  4          END SETUP$INPUT;
01802  3
01803  3
01804  3          GET$FIELD: PROCEDURE;
01805  4
01806  4               GEN$READ: PROCEDURE(I,J);
01807  5                    DECLARE (I,J) BYTE;
01808  5                    IF STYPESP = STRING THEN
01809  5                         DO;
01810  5                              CALL GENERATE(I);
01811  6                              CALL GENERATE(STS);
01812  6                         END;
01813  5                    ELSE
01814  5                         DO;
01815  5                              CALL GENERATE(J);
01816  6                              CALL GENERATE(STD);
01817  6                         END;
01818  5                    RETURN;
01819  5               END GEN$READ;
01820  4
01821  4               IF(TYPESP = SIMVAR) THEN
01822  4                    CALL LITERAL(SYMLOCSP);
01823  4               IF INPUTSTMT THEN
01824  4                    CALL GEN$READ(RES,RDV);
01825  4               ELSE
01826  4                    IF FILEIO THEN
01827  4                         CALL GEN$READ(RDS,RDN);
01828  4                    ELSE
01829  4                         CALL GEN$READ(DRS,DRF);
01830  4               RETURN;
01831  4          END GET$FIELD;
01832  3
01833  3
01834  3          GEN$ON: PROCEDURE;
01835  4               CALL GENERATE(RON);
01836  4               CALL LITERAL(ONSTACK(ONSTACK := ONSTACK + 1));
01837  4               CALL GENERATE(CKO);
01838  4               CALL GENERATE(BFN);
01839  4               RETURN;
01840  4          END GEN$ON;
01841  3
01842  3
01843  3          GEN$ON$2: PROCEDURE;
01844  4               ONSTACK(ONSTACK) = TYPESP;
01845  4               RETURN;
01846  4          END GEN$ON$2;
01847  3
01848  3
01849  3          GENNEXT: PROCEDURE;
01850  4               IF(FORCOUNT := FORCOUNT - 1) = 255 THEN
01851  4                    DO;
01852  4                         FORCOUNT = 0;
01853  5                         CALL ERROR('NU');
```

```
01854  5                END;
01855  4           ELSE
01856  4                DO;
01857  4                     CALL GENERATE(BRS);
01858  5                     CALL GEN$TWO(NEXTADDRESS(2));
01859  5                     NEXTADDRESS = CODESIZE OR 8000H;
01860  5                     DO WHILE NEXTBYTE(1) > 127;
01861  5                          NEXTSTMTPTR = NEXTSTMTPTR + 8;
01862  6                          END;
01863  5                END;
01864  4           RETURN;
01865  4      END GENNEXT;
01866  3
01867  3
01868  3      GEN$NEXT$WITH$IDENT: PROCEDURE;
01869  4           IF LOOKUP$ONLY(MPPI) AND (BASE = NEXTADDRESS(3)) THEN
01870  4                CALL GENNEXT;
01871  4           ELSE
01872  4                CALL ERROR('NI');
01873  4           RETURN;
01874  4      END GEN$NEXT$WITH$IDENT;
01875  3
01876  3
01877  3      CHECK$UL$ERROR: PROCEDURE;
01878  4           IF ULERRORFLAG THEN
01879  4                CALL ERROR('UL');
01880  4           ULERRORFLAG = FALSE;
01881  4      END CHECK$UL$ERROR;
01882  3
01883  3
01884  3      FINDLABEL: PROCEDURE;
01885  4           IF NORMAL$LOOKUP(SP) THEN
01886  4                DO;
01887  4                     IF PASS2 AND (NOT GETRES) THEN
01888  5                          ULERRORFLAG = TRUE;
01889  5                END;
01890  4           RETURN;
01891  4      END FINDLABEL;
01892  3
01893  3
01894  3       RESOLVE$LABEL: PROCEDURE;
01895  4           CALL FINDLABEL;
01896  4           IF GOSUBSTMT THEN
01897  4                CALL GENERATE(PRO);
01898  4           ELSE
01899  4                CALL GENERATE(BRS);
01900  4           CALL GEN$TWO(GETADDR);
01901  4           RETURN;
01902  4       END RESOLVE$LABEL;
01903  3
01904  3
01905  3      PROCESS$SIMPLE$VARIABLE: PROCEDURE(LOC);
01906  4           DECLARE LOC BYTE;
01907  4           IF NORMALLOOKUP(LOC) THEN
01908  4                DO;
01909  4                     IF GETYPE <> SIMVAR THEN
01910  5                          CALL ERROR('IU');
01911  5                END;
01912  4           ELSE
01913  4                DO;
01914  4                     CALL SETADDR(COUNTPRT);
01915  5                     CALL SETYPE(SIMVAR);
01916  5                END;
01917  4           CALL SETSYMLOCSP(GETADDR);
01918  4           CALL SETTYPESP(SIMVAR);
01919  4           IF FORSTMT THEN
01920  4                DO;
01921  4                     FORSTMT = FALSE;
01922  5                     FORADDRESS(3) = BASE;
01923  5                END;
01924  4      END PROCESS$SIMPLE$VARIABLE;
01925  3
01926  3
01927  3      GEN$ILS: PROCEDURE(WHERE);
01928  4            DECLARE STRPTR BYTE,
01929  4                 WHERE ADDRESS,
01930  4                 STRINGTOSPOOL BASED WHERE BYTE;
01931  4           CALL SETSTYPESP(STRING);
01932  4           CALL GENERATE(ILS);
01933  4           DO FOREVER;
01934  4                DO STRPTR = 1 TO STRINGTOSPOOL;
01935  5                     CALL GENERATE(STRINGTOSPOOL(STRPTR));
01936  6                     END;
01937  5                IF CONT THEN
01938  5                     CALL SCANNER;
01939  5                ELSE
01940  5                     DO;
01941  5                          CALL GENERATE(0);
01942  6                          RETURN;
01943  6                     END;
01944  5           END; /* OF DO FOREVER */
01945  4      END GEN$ILS;
01946  3
01947  3
01948  3      GENCON: PROCEDURE;
01949  4           DECLARE I BYTE;
01950  4           CALL GENERATE(CON);
01951  4           CALL SETTYPESP(CONST);
```

```
01952  4            CALL SETSTYPESP(FLOATPT);
01953  4            IF LCCKUP$ONLY(SP) AND (GETYPE = CONST) THEN
01954  4                 CALL GEN$TWO(GETADDR);
01955  4            ELSE
01956  4                 DO;
01957  4                      DO I = 1 TO ACCUM;
01958  5                           CALL EMITCON(ACCUM(I));
01959  6                           END;
01960  5                      CALL EMITCON('$');
01961  5                      CALL GEN$TWO(FDACT := FDACT + 1);
01962  5                 END;
01963  4            RETURN;
01964  4       END GENCCN;
01965  3
01966  3
01967  3       PUT$FIELD: PROCEDURE;
01968  4            IF FILEIO THEN
01969  4                 DO;
01970  4                      IF STYPESP = FLOATPT THEN
01971  5                           CALL GENERATE(WRN);
01972  5                      ELSE
01973  5                           CALL GENERATE(WRS);
01974  5                 END;
01975  4            ELSE
01976  4                 IF STYPESP = FLOATPT THEN
01977  4                      DO;
01978  4                           IF TYPESP <> 74 THEN /* IS IT A TAB */
01979  5                                CALL GENERATE(WRV);
01980  5                       END;
01981  4                 ELSE
01982  4                      CALL GENERATE(WST);
01983  4            RETURN;
01984  4        END PUT$FIELD;
01985  3
01986  3
01987  3       GEN$PARM: PROCEDURE;
01988  4            IF TYPEMP = UNFUNC THEN
01989  4                 DO;
01990  4                      BASE = SYMLOCMP;
01991  5                      CALL NEXTENTRY;
01992  5                      CALL SETSYMLOCMP(BASE);
01993  5                      CALL SETHASHMP(HASHMP := HASHMP - 1);
01994  5                      CALL LITERAL(GETADDR);
01995  5                 END;
01996  4            RETURN;
01997  4       END GEN$PARM;
01998  3
01999  3
02000  3       CHECKPARM: PROCEDURE;
02001  4            IF TYPEMP = UNFUNC THEN
02002  4                 DO;
02003  4                      BASE = SYMLOCMP;
02004  5                      IF(GETSUBTYPE <> STYPEMP1) THEN
02005  5                           CALL ERROR('FP');
02006  5                      CALL GEN$STORE;
02007  5                      RETURN;
02008  5                 END;
02009  4            IF(HASHMP XOR (STYPEMP1 <> FLOATPT)) THEN
02010  4                 CALL ERROR('FP');
02011  4            CALL SETHASHMP(SHR(HASHMP,1));
02012  4            CALL SETSTYPEMP(STYPEMP := STYPEMP -1);
02013  4            RETURN;
02014  4       END CHECKPARM;
02015  3
02016  3
02017  3       FUNCGEN: PROCEDURE;
02018  4            IF TYPEMP = UNFUNC THEN
02019  4                 DO;
02020  4                      IF HASHMP <> 0 THEN
02021  5                           CALL ERROR('FN');
02022  5                      CALL GENERATE(PRO);
02023  5                      BASE = SRLOCSP;
02024  5                      CALL GEN$TWO(GETADDR);
02025  5                      RETURN;
02026  5                 END;
02027  4            IF((STYPEMP AND 03H) <>0) THEN
02028  4                 CALL ERROR('FN');
02029  4            CALL GENERATE(TYPEMP);
02030  4            IF ROL(STYPEMP,2) THEN
02031  4                 CALL SETSTYPEMP(STRING);
02032  4            ELSE
02033  4                 CALL SETSTYPEMP(FLOATPT);
02034  4            RETURN;
02035  4       END FUNCGEN;
02036  3
02037  3
02038  3       ENTER$PARM: PROCEDURE;
02039  4            IF PASS1 THEN
02040  4                 DO;
02041  4                      CALL SETLOOKUP(MPP1);
02042  5                      CALL ENTER;
02043  5                      CALL SETADDR(COUNTPRT);
02044  5                      CALL SETSUBTYPE(STYPEMP1);
02045  5                      CALL SETYPE(SIMVAR);
02046  5                      CALL SETTYPEMP(TYPEMP + 1);
02047  5                 END;
02048  4            RETURN;
02049  4       END ENTER$PARM;
```

```
02050  3
02051  3          /*
02052  3           ************************************************************
02053  3           *                                                          *
02054  3           *     EXECUTION OF SYNTHESIS BEGINS HERE.....              *
02055  3           *                                                          *
02056  3           ************************************************************
02057  3          */
02058  3
02059  3      IF LISTPROD AND PASS2 THEN
02060  3          DO;  /*  IF LISTPROD SET PRINT OUT PRODUCTIONS */
02061  3               CALL PRINT(.'PROD $');
02062  4               CALL PRINTDEC(PRODUCTION);
02063  4               CALL CRLF;
02064  4          END;
02065  3      CALL COPY;  /*  SETUP FOR ACCESSING PARSE TABLES */
02066  3      DO CASE PRODUCTION;  /* CALL TO SYNTHESIS HANDLES ONE PROD */
02067  3      /* CASE 0 NOT USED */ ;
02068  4  /*       1   <PROGRAM> ::= <LINE NUMBER> <STATEMENT>                */
02069  4               ;
02070  4  /*       2   <LINE NUMBER> ::= <NUMBER>                             */
02071  4          DO;
02072  4               IF LOOKUP$ONLY(SP) THEN
02073  5                    DO;
02074  5                         IF GETRES THEN
02075  6                              DO;
02076  6                                   IF CODESIZE <> GETADDR THEN
02077  7                                        CALL ERROR('DL');
02078  7                              END;
02079  6                         ELSE
02080  6                              DO;
02081  6                                   CALL SETADDR(CODESIZE);
02082  7                                   CALL SETYPE(LABLE);
02083  7                              END;
02084  6                    END;
02085  5               ELSE
02086  5                    SEPARATOR = ASTRICK;
02087  5               CALL LINE$NUMBER;
02088  5          END;
02089  4  /*       3                                                          */
02090  4          CALL LINE$NUMBER;
02091  4  /*       4   <STATEMENT> ::= <STATEMENT LIST>                       */
02092  4           CALL CHECK$UL$ERROR;
02093  4  /*       5                   <IF STATEMENT>                         */
02094  4               ;
02095  4  /*       6                   <END STATEMENT>                        */
02096  4               ;
02097  4  /*       7                   <DIMENSION STATEMENT>                  */
02098  4               ;
02099  4  /*       8                   <DEFINE STATEMENT>                     */
02100  4               ;
02101  4  /*       9   <STATEMENT LIST> ::= <SIMPLE STATEMENT>                */
02102  4               ;
02103  4  /*      10                        <STATEMENT LIST> :                */
02104  4  /*      10                        <SIMPLE STATEMENT>                */
02105  4               ;
02106  4  /*      11   <SIMPLE STATEMENT> ::= <LET STATEMENT>                 */
02107  4               ;
02108  4  /*      12                          <ASSIGNMENT>                    */
02109  4               ;
02110  4  /*      13                          <FOR STATEMENT>                 */
02111  4               ;
02112  4  /*      14                          <NEXT STATEMENT>                */
02113  4               ;
02114  4  /*      15                          <FILE STATEMENT>                */
02115  4               ;
02116  4  /*      16                          <CLOSE STATEMENT>               */
02117  4               ;
02118  4  /*      18                          <PRINT STATEMENT>               */
02119  4  /*      17                          <READ STATEMENT>                */
02120  4               ;
02121  4               ;
02122  4  /*      19                          <GOTO STATEMENT>                */
02123  4               ;
02124  4  /*      20                          <GOSUB STATEMENT>               */
02125  4               ;
02126  4  /*      21                          <INPUT STATEMENT>               */
02127  4               ;
02128  4  /*      22                          <STOP STATEMENT>                */
02129  4               ;
02130  4  /*      23                          <RETURN STATEMENT>              */
02131  4               ;
02132  4  /*      24                          <ON STATEMENT>                  */
02133  4               ;
02134  4  /*      25                          <RESTORE STATEMENT>             */
02135  4               ;
02136  4  /*      26                          <RANDOMIZE STATEMENT>           */
02137  4               ;
02138  4  /*      27                          <OUT STATEMENT>                 */
02139  4               ;
02140  4  /*      28                                                          */
02141  4               ;
02142  4  /*      29   <LET STATEMENT> ::= LET <ASSIGNMENT>                   */
02143  4               ;
02144  4  /*      30   <ASSIGNMENT> ::= <ASSIGN HEAD> <EXPRESSION>            */
02145  4           IF CHKTYP2 THEN
02146  4               CALL GEN$STORE;
02147  4  /*      31   <ASSIGN HEAD> ::= <VARIABLE> =                         */
```

```
02148  4            IF TYPEMP = SIMVAR THEN
02149  4              CALL LITERAL(SYMLOCMP);
02150  4    /*   32   <EXPRESSION> ::= <LOGICAL FACTOR>                          */
02151  4            ;
02152  4    /*   33                    <EXPRESSION> <OR> <LOGICAL FACTOR>        */
02153  4            IF CHKTYP1 THEN
02154  4              CALL GENERATE(TYPEMP1);
02155  4    /*   34   <OR> ::= OR                                                */
02156  4            CALL SETTYPESP(BOR);
02157  4    /*   35            XOR                                               */
02158  4            CALL SETTYPESP(EXR);
02159  4    /*   36   <LOGICAL FACTOR> ::= <LOGICAL SECONDARY>                   */
02160  4            ;
02161  4    /*   37                        <LOGICAL FACTOR> AND                  */
02162  4    /*   37                        <LOGICAL SECONDARY>                   */
02163  4            IF CHKTYP1 THEN
02164  4              CALL GENERATE(ANDO);
02165  4    /*   38   <LOGICAL SECONDARY> ::= <LOGICAL PRIMARY>                  */
02166  4            ;
02167  4    /*   39                           NOT <LOGICAL PRIMARY>              */
02168  4            IF CHKTYP3 THEN
02169  4              CALL GENERATE(NOTO);
02170  4    /*   40   <LOGICAL PRIMARY> ::= <ARITHMETIC EXPRESSION>              */
02171  4            ;
02172  4    /*   41                         <ARITHMETIC EXPRESSION>              */
02173  4    /*   41                         <RELATION>                           */
02174  4    /*   41                         <ARITHMETIC EXPRESSION>              */
02175  4            IF CHKTYP2 THEN
02176  4              DO;
02177  4                   IF STYPESP = FLOATPT THEN
02178  5                        CALL GENERATE(TYPEMP1);
02179  5                   ELSE
02180  5                        DO;
02181  5                             CALL GENERATE(TYPEMP1 + 16);
02182  6                             CALL SETSTYPEMP(FLOATPT);
02183  6                        END;
02184  5              END;
02185  4    /*   42   <ARITHMETIC EXPRESSION> ::= <TERM>                         */
02186  4            ;
02187  4    /*   43                                <ARITHMETIC EXPRESSION> +     */
02188  4    /*   43                                <TERM>                        */
02189  4            IF CHKTYP2 THEN
02190  4              DO;
02191  4                   IF STYPESP = FLOATPT THEN
02192  5                        CALL GENERATE(FAD);
02193  5                   ELSE
02194  5                        CALL GENERATE(CAT);
02195  5              END;
02196  4    /*   44                                <ARITHMETIC EXPRESSION> -     */
02197  4    /*   44                                <TERM>                        */
02198  4            IF CHKTYP1 THEN
02199  4              CALL GENERATE(FMI);
02200  4    /*   45                                + <TERM>                      */
02201  4            IF CHKTYP3 THEN   ; /* NO ACTION REQUIRED */
02202  4    /*   46                                - <TERM>                      */
02203  4            IF CHKTYP3 THEN
02204  4              CALL GENERATE(NEG);
02205  4    /*   47   <TERM> ::= <PRIMARY>                                       */
02206  4            ;
02207  4    /*   48                <TERM> * <PRIMARY>                            */
02208  4            IF CHKTYP1 THEN
02209  4              CALL GENERATE(FMU);
02210  4    /*   49                <TERM> / <PRIMARY>                            */
02211  4            IF CHKTYP1 THEN
02212  4              CALL GENERATE(FDI);
02213  4    /*   50   <PRIMARY> ::= <ELEMENT>                                    */
02214  4            ;
02215  4    /*   51                <PRIMARY> ** <ELEMENT>                        */
02216  4            IF CHKTYP1 THEN
02217  4              CALL GENERATE(EXP);
02218  4    /*   52   <ELEMENT> ::= <VARIABLE>                                   */
02219  4            IF TYPESP = SIMVAR THEN
02220  4              CALL LITLOAD(SYMLOCSP);
02221  4            ELSE
02222  4              CALL GENERATE(LOD);
02223  4    /*   53                <CONSTANT>                                    */
02224  4            ;
02225  4    /*   54                <FUNCTION CALL>                               */
02226  4            ;
02227  4    /*   55                ( <EXPRESSION> )                              */
02228  4            CALL SETSTYPEMP(STYPEMP1);
02229  4    /*   56   <VARIABLE> ::= <IDENTIFIER>                                */
02230  4            CALL PROCESS$SIMPLE$VARIABLE(SP);
02231  4    /*   57                  <SUBSCRIPT HEAD> <EXPRESSION> )             */
02232  4            DO;
02233  4                 IF FORSTMT THEN
02234  5                      CALL ERROR('FI');
02235  5                 CALL CHKTYP4;
02236  5                 BASE = SYMLOCMP;
02237  5                 IF GETSUBTYPE <> TYPEMP THEN
02238  5                      CALL ERROR('SN');
02239  5                 CALL LITLOAD(GETADDR);
02240  5                 CALL GENERATE(SUBO);
02241  5                 CALL SETTYPEMP(SUBVAR);
02242  5            END;
02243  4    /*   58   <SUBSCRIPT HEAD> ::= <IDENTIFIER> (                        */
02244  4            DO;
02245  4                 IF((NOT LOOKUP$ONLY(MP)) OR (GETYPE <> SUBVAR)) THEN
02246  5                      CALL ERROR('IS');
```

```
02247  5            CALL SETTYPEMP(0);
02248  5            CALL SETSYMLOCMP(BASE);
02249  5            END;
02250  4     /*     59                          <SUBSCRIPT HEAD> <EXPRESSION> ,      */
02251  4            CALL CHKTYP4;
02252  4     /*     60    <FUNCTION CALL> ::= <FUNCTION HEADING> <EXPRESSION> )    */
02253  4            DO;
02254  4                  CALL CHECKPARM;
02255  5                  SRLOCSP = SRLOCMP;
02256  5                  CALL FUNCGEN;
02257  5            END;
02258  4     /*     61                          <FUNCTION NAME>                      */
02259  4            CALL FUNCGEN;
02260  4     /*     62    <FUNCTION HEADING> ::= <FUNCTION NAME> (                  */
02261  4            CALL GEN$PARM;
02262  4     /*     63                             <FUNCTION HEADING> <EXPRESSION>   */
02263  4     /*     63                             ,                                 */
02264  4            DO;
02265  4                  CALL CHECK$PARM;
02266  5                  CALL GEN$PARM;
02267  5            END;
02268  4     /*     64    <FUNCTION NAME> ::= <USERDEFINED NAME>                    */
02269  4            IF LOCKUP$ONLY(SP) THEN
02270  4                  DO;
02271  4                        CALL SETSRLOCSP(BASE);
02272  5                        CALL SETSYMLOCSP(BASE);
02273  5                        CALL SETTYPESP(UNFUNC);
02274  5                        CALL SETHASHSP(GETYPE);
02275  5                  END;
02276  4            ELSE
02277  4                  CALL ERROR('FU');
02278  4     /*     65                          <PREDEFINED NAME>                    */
02279  4            DO;
02280  4                  CALL SETTYPESP(FUNCOP);
02281  5                  CALL SETHASHSP(SHR(STYPESP,2) AND 07H);
02282  5             END;
02283  4     /*     66    <CONSTANT> ::= <NUMBER>                                   */
02284  4            CALL GENCON;
02285  4     /*     67                       <STRING>                               */
02286  4            CALL GEN$ILS(.ACCUM);
02287  4     /*     68    <RELATION> ::= =                                          */
02288  4            CALL SETTYPESP(7);
02289  4     /*     69                   > =                                       */
02290  4            CALL SETTYPEMP(9);
02291  4     /*     70                   GE                                        */
02292  4            CALL SETTYPEMP(9);
02293  4     /*     71                   < =                                       */
02294  4            CALL SETTYPEMP(10);
02295  4     /*     72                   LE                                        */
02296  4            CALL SETTYPEMP(10);
02297  4     /*     73                   >                                         */
02298  4            CALL SETTYPESP(6);
02299  4     /*     74                   <                                         */
02300  4            CALL SETTYPESP(5);
02301  4     /*     75                   < >                                       */
02302  4            CALL SETTYPEMP(8);
02303  4     /*     76                   NE                                        */
02304  4            CALL SETTYPEMP(8);
02305  4     /*     77    <FOR STATEMENT> ::= <FOR HEAD> TO <EXPRESSION>            */
02306  4     /*     77                        <STEP CLAUSE>                         */
02307  4            DO;
02308  4                  BASE = FORADDRESS(3);
02309  5                  IF TYPESP THEN
02310  5                        CALL GENERATE(DUP);
02311  5                  CALL LITLOAD(GETADDR);
02312  5                  CALL GENERATE(FAD);
02313  5                   IF TYPESP THEN
02314  5                         DO;
02315  5                               CALL LITERAL(GETADDR);
02316  6                               CALL GENERATE(XCH);
02317  6                         END;
02318  5                  CALL GENERATE(STO);
02319  5                  IF TYPESP THEN
02320  5                        DO;
02321  5                              CALL GENERATE(XCH);
02322  6                              CALL LITERAL(0);
02323  6                              CALL GENERATE(LSS);
02324  6                              CALL LITERAL(5);
02325  6                              CALL GENERATE(BFC);
02326  6                              CALL GENERATE(LEQ);
02327  6                              CALL LITERAL(2);
02328  6                              CALL GENERATE(BFN);
02329  6                        END;
02330  5                  CALL GENERATE(GEQ);
02331  5                  CALL GENERATE(BRC);
02332  5                  CALL GEN$TWO(FORADDRESS);
02333  5                  FORADDRESS(1) = CODESIZE;
02334  5            END;
02335  4     /*     78    <FOR HEAD> ::= <FOR> <ASSIGNMENT>                         */
02336  4            DO;
02337  4                  CALL GENERATE(BRS);
02338  5                  CALL GEN$TWO(FORADDRESS(1));
02339  5                  FORADDRESS(2) = CODESIZE;
02340  5            END;
02341  4     /*     79    <FOR> ::= FOR                                             */
02342  4            DO;
02343  4                  FORSTMT = TRUE;
02344  5                  SBTBLTOP.NEXTSTMTPTR = SBTBLTOP - 8;
```

```
02345  5                NEXTBYTE(1) = NEXTBYTE(1) AND 7FH;
02346  5                CALL LIMITS(0);
02347  5                FORCOUNT = FORCOUNT + 1;
02348  5             END;
02349  4     /*      80   <STEP CLAUSE> ::= STEP <EXPRESSION>                    */
02350  4             CALL SETTYPEMP(TRUE);
02351  4     /*      81                                                          */
02352  4             DO;
02353  4                BASE = FORADDRESS(3);
02354  5                CALL LITERAL(GETADDR);
02355  5                CALL SETTYPESP(FALSE);
02356  5                CALL GENERATE(CON);
02357  5                CALL GEN$TWO(0);
02358  5             END;
02359  4     /*      82   <IF STATEMENT> ::= <IF GROUP>                          */
02360  4             CALL ENTER$COMPILER$LABEL(0);
02361  4     /*      83                      <IF ELSE GROUP> <STATEMENT LIST>    */
02362  4             CALL ENTER$COMPILER$LABEL(0);
02363  4     /*      84                      IF END   <EXPRESSION> THEN <NUMBER> */
02364  4             DO;
02365  4                CALL GENERATE(DEF);
02366  5                CALL FINDLABEL;
02367  5                CALL GEN$TWO(GETADDR);
02368  5             END;
02369  4     /*      85   <IF GROUP> ::= <IF HEAD> <STATEMENT LIST>              */
02370  4                ;
02371  4     /*      86                  <IF HEAD> <NUMBER>                      */
02372  4             CALL RESOLVE$LABEL;
02373  4     /*      87   <IF ELSE GROUP> ::= <IF HEAD> <STATEMENT LIST> ELSE    */
02374  4             DO;
02375  4             CALL ENTER$COMPILER$LABEL(3);
02376  5                CALL GENERATE(BRS);
02377  5                CALL COMPILER$LABEL;
02378  5             END;
02379  4     /*      88   <IF HEAD> ::= IF <EXPRESSION> THEN                     */
02380  4             DO;
02381  4                IF STYPEMP1 = STRING THEN
02382  5                   CALL ERROR('IE');
02383  5                CALL GENERATE(BRC);
02384  5                CALL COMPILER$LABEL;
02385  5             END;
02386  4     /*      89   <DEFINE STATEMENT> ::= <UD FUNCTION NAME>              */
02387  4     /*      89                          <DUMMY ARG LIST> = <EXPRESSION> */
02388  4             IF CHKTYP2 THEN
02389  4                DO;
02390  4                   BASE = SYMLOCMP;
02391  5                   CALL SETYPE(TYPEMP1);
02392  5                   CALL UNLINK;
02393  5                   CALL GENERATE(XCH);
02394  5                   CALL GENERATE(RTN);
02395  5                   CALL ENTER$COMPILER$LABEL(0);
02396  5                END;
02397  4     /*      90   <UD FUNCTION NAME> ::= DEF <USERDEFINED NAME>          */
02398  4             DO;
02399  4                DECLARE FLAG BYTE;
02400  5                CALL GENERATE(BRS);
02401  5                CALL COMPILER$LABEL;
02402  5                FLAG = NORMAL$LOOKUP(SP);
02403  5                CALL SETSTYPEMP(STYPESP);
02404  5                CALL SETSYMLOCMP(BASE);
02405  5                IF PASS1 THEN
02406  5                   DO;
02407  5                      IF FLAG THEN
02408  6                         CALL ERROR('FD');
02409  6                      CALL SETADDR(CODESIZE);
02410  6                   END;
02411  5                ELSE
02412  5                   CALL RELINK;
02413  5             END;
02414  4     /*      91   <DUMMY ARG LIST> ::= <DUMMY ARG HEAD> <IDENTIFIER> )   */
02415  4             CALL ENTER$PARM;
02416  4     /*      92                                                          */
02417  4             CALL SETTYPEMP(0);
02418  4     /*      93   <DUMMY ARG HEAD> ::= (                                 */
02419  4             CALL SETTYPEMP(0);
02420  4     /*      94                        <DUMMY ARG HEAD> <IDENTIFIER> ,   */
02421  4             CALL ENTER$PARM;
02422  4     /*      95   <FILE STATEMENT> ::= <FILE HEAD> <FILE DECLERATION>    */
02423  4                ;
02424  4     /*      96   <FILE HEAD> ::= FILE                                   */
02425  4                ;
02426  4     /*      97                    <FILE HEAD> <FILE DECLERATION> ,      */
02427  4                ;
02428  4     /*      98   <FILE DECLERATION> ::= <IDENTIFIER> <FILE REC SIZE>    */
02429  4             DO;
02430  4                CALL PROCESS$SIMPLE$VARIABLE(MP);
02431  5                IF TYPESP = FLOATPT THEN
02432  5                   CALL ERROR('IF');
02433  5                CALL LITLOAD(SYMLOCSP);
02434  5                CALL GENERATE(OPN);
02435  5             END;
02436  4     /*      99   <FILE REC SIZE> ::= ( <EXPRESSION> )                   */
02437  4             CALL CHKTYP4;
02438  4     /*     100                                                          */
02439  4             CALL LITERAL(0);
02440  4     /*     101   <DIMENSION STATEMENT> ::= DIM                          */
02441  4     /*     101                             <DIMENSION VARIABLE LIST>    */
02442  4                ;
```

```
02443  4     /*    102   <DIMENSION VARIABLE LIST> ::= <DIMENSION VARIABLE>        */
02444  4       CALL SUBCALC;
02445  4     /*    103                                                           */
02446  4     /*    103                                    <DIMENSION VARIABLE LIST>  */
02447  4     /*    103                                     , <DIMENSION VARIABLE>    */
02448  4       CALL SUBCALC;
02449  4     /*    104   <DIMENSION VARIABLE> ::= <DIM VAR HEAD> <EXPRESSION> )  */
02450  4       DO;
02451  4             CALL CHKTYP4;
02452  5             BASE = SYMLOCMP;
02453  5       END;
02454  4     /*    105   <DIM VAR HEAD> ::= <IDENTIFIER> (                       */
02455  4             DO;
02456  4                   IF NORMAL$LOOKUP(MP) AND PASS1 THEN
02457  5                         CALL ERROR('DP');
02458  5                   CALL SETYPE(SUBVAR);
02459  5                   IF PASS1 THEN
02460  5                         CALL SETADDR(COUNTPRT);
02461  5                   CALL LITERAL(GETADDR);
02462  5                   CALL SETTYPEMP(0);
02463  5                   CALL SETSYMLOCMP(BASE);
02464  5             END;
02465  4     /*    106                         <DIM VAR HEAD> <EXPRESSION> ,     */
02466  4       CALL CHKTYP4;
02467  4     /*    107   <CLOSE STATEMENT> ::= CLOSE <CLOSE LIST>                */
02468  4       ;
02469  4     /*    108   <CLOSE LIST> ::= <EXPRESSION>                           */
02470  4       DO;
02471  4             CALL CHKTYP4;
02472  5             CALL GENERATE(CLS);
02473  5       END;
02474  4     /*    109                       <CLOSE LIST> , <EXPRESSION>         */
02475  4       DO;
02476  4             CALL CHKTYP4;
02477  5             CALL GENERATE(CLS);
02478  5       END;
02479  4     /*    110   <READ STATEMENT> ::= READ <FILE OPTION> <READ LIST>     */
02480  4       IF FILEIO THEN
02481  4             DO;
02482  4                   CALL GENERATE(EDR);
02483  5                   FILEIO = FALSE;
02484  5             END;
02485  4     /*    111                             READ <READ LIST>              */
02486  4       ;
02487  4     /*    112   <INPUT STATEMENT> ::= INPUT <PROMPT OPTION>             */
02488  4     /*    112                         <READ LIST>                       */
02489  4       DO;
02490  4             CALL GENERATE(ECR);
02491  5             INPUTSTMT = FALSE;
02492  5       END;
02493  4     /*    113   <PROMPT OPTION> ::= <CONSTANT> ;                        */
02494  4       DO;
02495  4             CALL PUT$FIELD;
02496  5             CALL SETUP$INPUT;
02497  5       END;
02498  4     /*    114                                                           */
02499  4       CALL SETUP$INPUT;
02500  4     /*    115   <READ LIST> ::= <VARIABLE>                              */
02501  4       CALL GET$FIELD;
02502  4     /*    116                       <READ LIST> , <VARIABLE>            */
02503  4       CALL GET$FIELD;
02504  4     /*    117                                                           */
02505  4       FILEIO = FALSE;
02506  4     /*    118   <PRINT STATEMENT> ::= PRINT <PRINT LIST> <PRINT END>    */
02507  4       ;
02508  4     /*    119                             PRINT <FILE OPTION> <FILE LIST>  */
02509  4       DO;
02510  4             CALL GENERATE(EOW);
02511  5             FILEIO = FALSE;
02512  5       END;
02513  4     /*    120   <PRINT LIST> ::= <EXPRESSION>                           */
02514  4       CALL PUT$FIELD;
02515  4     /*    121                       <PRINT LIST> <PRINT DELIM>          */
02516  4     /*    121                       <EXPRESSION>                        */
02517  4       CALL PUT$FIELD;
02518  4     /*    122                                                           */
02519  4       ;
02520  4     /*    123   <FILE LIST> ::= <EXPRESSION>                            */
02521  4       CALL PUT$FIELD;
02522  4     /*    124                       <EXPRESSION> , <EXPRESSION>         */
02523  4       CALL PUT$FIELD;
02524  4     /*    125   <PRINT END> ::= <PRINT DELIM>                           */
02525  4       ;
02526  4     /*    126                                                           */
02527  4       CALL GENERATE(DBF);
02528  4     /*    127   <FILE OPTION> ::=   <EXPRESSION> ;                      */
02529  4       DO;
02530  4             FILEIO = TRUE;
02531  5             CALL GENERATE(RON);
02532  5             CALL GENERATE(RDB);
02533  5       END;
02534  4     /*    128                           <EXPRESSION> , <EXPRESSION> ;   */
02535  4       DO;
02536  4             FILEIO = TRUE;
02537  5             CALL GENERATE(RON);
02538  5             CALL GENERATE(XCH);
02539  5             CALL GENERATE(RON);
02540  5             CALL GENERATE(RDF);
```

```
02541  5          END;
02542  4    /*    129   <PRINT DELIM> ::= ;                                          */
02543  4               ;
02544  4    /*    130                   ,                                          */
02545  4          IF NOT FILEIO THEN
02546  4               CALL GENERATE(NSP);
02547  4    /*    131   <GOTO STATEMENT> ::= <GOTO> <NUMBER>                       */
02548  4          CALL RESOLVE$LABEL;
02549  4    /*    132   <ON STATEMENT> ::= <ON GOTO> <LABEL LIST>                  */
02550  4          CALL GEN$ON$2;
02551  4    /*    133                      <ON GOSUB> <LABEL LIST>                 */
02552  4          DO;
02553  4               CALL GEN$ON$2;
02554  5               CALL ENTER$COMPILER$LABEL(0);
02555  5          END;
02556  4    /*    134   <ON GOTO> ::= ON <EXPRESSION> <GOTO>                       */
02557  4          CALL GEN$ON;
02558  4    /*    135   <ON GOSUB> ::= ON <EXPRESSION> <GOSUB>                     */
02559  4          DO;
02560  4               CALL SET$COMPILER$LABEL;
02561  5               CALL LITERAL(GETADDR);
02562  5                CALL GENERATE(ADJ);
02563  5                CALL GENERATE(XCH);
02564  5               CALL GEN$ON;
02565  5          END;
02566  4    /*    136   <LABEL LIST> ::= <NUMBER>                                  */
02567  4          DO;
02568  4               CALL RESOLVE$LABEL;
02569  5               CALL SETTYPESP(1);
02570  5          END;
02571  4    /*    137                      <LABEL LIST> , <NUMBER>                 */
02572  4          DO;
02573  4               CALL RESOLVE$LABEL;
02574  5               CALL SETTYPEMP(TYPEMP + 1);
02575  5          END;
02576  4    /*    138   <GOSUB STATEMENT> ::= <GOSUB> <NUMBER>                     */
02577  4          DO;
02578  4               GOSUBSTMT = TRUE;
02579  5               CALL RESOLVE$LABEL;
02580  5               GOSUBSTMT = FALSE;
02581  5          END;
02582  4    /*    139   <GOTO> ::= GOTO                                            */
02583  4               ;
02584  4    /*    140              GO TO                                           */
02585  4               ;
02586  4    /*    141   <GOSUB> ::= GOSUB                                          */
02587  4               ;
02588  4    /*    142               GO SUB                                         */
02589  4               ;
02590  4    /*    143   <NEXT STATEMENT> ::= <NEXT HEAD> <IDENTIFIER>              */
02591  4          CALL GEN$NEXT$WITH$IDENT;
02592  4    /*    144                        NEXT                                  */
02593  4          CALL GENNEXT;
02594  4    /*    145   <NEXT HEAD> ::= NEXT                                       */
02595  4               ;
02596  4    /*    146                   <NEXT HEAD> <IDENTIFIER> ,                 */
02597  4          CALL GEN$NEXT$WITH$IDENT;
02598  4    /*    147   <OUT STATEMENT> ::= OUT <EXPRESSION> , <EXPRESSION>        */
02599  4          IF STYPEMP1 <> FLOATPT OR STYPESP <> FLOATPT THEN
02600  4               CALL ERROR('MF');
02601  4          ELSE
02602  4               CALL GENERATE(POT);
02603  4    /*    148   <RETURN STATEMENT> ::= RETURN                              */
02604  4          CALL GENERATE(RTN);
02605  4    /*    149   <STOP STATEMENT> ::= STOP                                  */
02606  4          CALL GENERATE(XIT);
02607  4    /*    150   <END STATEMENT> ::= END                                    */
02608  4          IF PASS1 THEN
02609  4               DO;
02610  4                    PASS1 = FALSE;
02611  5                    CALL REWIND$SOURCE$FILE;
02612  5                    IF FORCOUNT <> 0 THEN
02613  5                         DO;
02614  5                              CALL ERROR('FU');
02615  6                              FORCOUNT = 0;
02616  6                         END;
02617  5                    CALL GENERATE('*');
02618  5                    CALL GENTWO((CODESIZE + 3) AND OFFFCH);
02619  5                    CALL GENTWO(DATACT);
02620  5                    CALL GENTWO(COUNTPRT);
02621  5               END;
02622  4          ELSE
02623  4               DO;
02624  4                    DO WHILE NEXTCHAR <> EOLCHAR;
02625  5                         NEXTCHAR = GETCHAR;
02626  6                         END;
02627  5                    CALL GENERATE(XIT);
02628  5                    CALL GENERATE(7FH);
02629  5                    CALL WRITE$INT$FILE;
02630  5                    CALL CLOSE$INT$FILE;
02631  5                    CALL PRINTDEC(ERRORCOUNT);
02632  5                    CALL PRINT(.' ERRORS DETECTED$');
02633  5                    CALL CRLF;
02634  5                    CALL MON3;
02635  5               END;
02636  4    /*    151   <RESTORE STATEMENT> ::= RESTORE                            */
02637  4          CALL GENERATE(RST);
02638  4    /*    152   <RANDOMIZE STATEMENT> ::= RANDOMIZE                        */
02639  4          CALL GENERATE(IRN);
```

```
02640  4        END /* OF CASES */;
02641  3
02642  3        RETURN;
02643  3    END SYNTHESIZE;
02644  2
02645  2
02646  2
02647  2        DECLARE
02648  2                I       INDEXSIZE,
02649  2                J       INDEXSIZE,
02650  2                K       INDEXSIZE,
02651  2                INDEX   BYTE;
02652  2
02653  2        INITIALIZE: PROCEDURE;
02654  3            CALL INITIALIZE$SYMTBL;
02655  3            CALL INITIALIZE$SYNTHESIZE;
02656  3            CALL INITIALIZE$SCANNER;
02657  3            RETURN;
02658  3        END INITIALIZE;
02659  2
02660  2
02661  2        GETIN1: PROCEDURE INDEXSIZE;
02662  3            RETURN INDEX1(STATE);
02663  3         END GETIN1;
02664  2
02665  2
02666  2        GETIN2: PROCEDURE INDEXSIZE;
02667  3            RETURN INDEX2(STATE);
02668  3         END GETIN2;
02669  2
02670  2
02671  2         INCSP: PROCEDURE;
02672  3            IF (SP := SP + 1) = LENGTH(STATESTACK) THEN
02673  3                CALL ERROR('SO');
02674  3            RETURN;
02675  3         END INCSP;
02676  2
02677  2
02678  2         LOOKAHEAD: PROCEDURE;
02679  3             IF NOLOOK THEN
02680  3                 DO;
02681  3                     CALL SCANNER;
02682  4                     NOLOOK = FALSE;
02683  4                 END;
02684  3            RETURN;
02685  3         END LOCKAHEAD;
02686  2
02687  2
02688  2         SET$VARC$I: PROCEDURE(I);
02689  3             DECLARE I BYTE;
02690  3             /* SET VARC, AND INCREMENT VARINDEX */
02691  3             VARC(VARINDEX)=I;
02692  3            IF(VARINDEX:=VARINDEX+1) > LENGTH(VARC) THEN
02693  3                CALL ERROR('VO');
02694  3            RETURN;
02695  3         END SET$VARC$I;
02696  2
02697  2            /*
02698  2             ************************************************************
02699  2             *                                                          *
02700  2             *          EXECUTION OF THE COMPILER BEGINS HERE           *
02701  2             *                                                          *
02702  2             *          THE OUTPUT FILE IS CREATED AND THE              *
02703  2             *          SYMBOLTABLE, SYNTHESIZE AND SCANNER             *
02704  2             *          ARE INITIALIZED.  THEN THE PARSER               *
02705  2             *          BEGINS PROCESSING THE SOURCE PROGRAM.           *
02706  2             *          PROCESSING CONTINUES UNTIL AN END               *
02707  2             *          STATEMENT IS INCOUNTERED OR UNTIL THE           *
02708  2             *          END OF THE SOURCE FILE IS DETECTED.             *
02709  2             *          AT THIS TIME THE THREE MAIN PROCEDURES          *
02710  2             *          ARE INITIALIZED FOR PASS 2 AND THE              *
02711  2             *          PARSER PROCESSES THE SOURCE FILE A              *
02712  2             *          SECOND TIME. AT THE END OF EACH STATE-          *
02713  2             *          MENT (WHICH TO THE PARSER IS A PROGRAM)         *
02714  2             *          AND IF AN ERROR IS DETECTED THE PARSER          *
02715  2             *          VARIABLES ARE REINITIALIZED BY SETTING          *
02716  2             *          COMPILING FALSE.                                *
02717  2             *                                                          *
02718  2             ************************************************************
02719  2            */
02720  2
02721  2        CALL PRINT(.'NBASIC COMPILER  VER 1.2$');
02722  2        CALL CRLF;
02723  2        CALL INITIALIZE;  /* INITIALIZE MAJOR SYSTEMS PRIOR TO PARSING */
02724  2
02725  2        DO FOREVER;  /* THIS LOOP CONTROLS THE 2 PASSES OF THE COMPILER */
02726  2        DO WHILE (PASS1 OR PASS2);/* THIS LOOP REINITIALIZES ON ERR OR OOC */
02727  3             /* INITIALIZE VARIABLES */
02728  3             COMPILING,NOLOCK=TRUE; STATE=STARTS;
02729  4             SP=255;
02730  4             VARINDEX,VAR = 0;
02731  4
02732  4             DO WHILE COMPILING;
02733  4                 IF STATE <= MAXRNO THEN  /* READ STATE */
02734  5                     DO;
02735  5                         CALL INCSP;
02736  6                         STATESTACK(SP) = STATE;
```

```
02737  6                         I = GETIN1;
02738  6                         CALL LOOKAHEAD;
02739  6                         J = I + GETIN2 - 1;
02740  6                         DO I = I TO J;
02741  6                              IF READ1(I) = TOKEN THEN  /* SAVE TOKEN */
02742  7                                   DO;
02743  7                                        VAR(SP) = VARINDEX;
02744  8                                        DO INDEX = 0 TO ACCUM;
02745  8                                             CALL SET$VARC$I(ACCUM(INDEX));
02746  9                                             END;
02747  8                                        HASH(SP) = HASHCODE;
02748  3                                        STYPE(SP) = SUBTYPE;
02749  8                                        STATE = READ2(I);
02750  8                                        NOLOOK = TRUE;
02751  8                                        I = J;
02752  8                                   END;
02753  7                              ELSE
02754  7                                   IF I = J THEN
02755  7                                        CALL ERROR('NP');
02756  7                         END;
02757  6                    END;
02758  5
02759  5               ELSE
02760  5                    IF STATE > MAXPNO THEN  /* APPLY PRODUCTION STATE */
02761  5                         DO;
02762  5                              MP = SP - GETIN2;
02763  6                              MPP1 = MP + 1;
02764  6                              CALL SYNTHESIZE(STATE - MAXPNO);
02765  6                              IF COMPILING THEN
02766  6                                   DO;
02767  6                                        SP = MP;
02768  7                                        I = GETIN1;
02769  7                                        VARINDEX = VAR(SP);
02770  7                                        J = STATESTACK(SP);
02771  7                                        DO WHILE (K := APPLY1(I)) <> 0
02772  7                                                  AND J <> K;
02773  7                                             I = I + 1;
02774  8                                             END;
02775  7                                        IF(STATE := APPLY2(I)) = 0 THEN
02776  7                                             COMPILING = FALSE;
02777  7                                   END;
02778  6                         END;
02779  5                    ELSE
02780  5                         IF STATE<= MAXLNO THEN /* LOOKAHEAD STATE */
02781  5                              DO;
02782  5                                   I = GETIN1;
02783  6                                   CALL LOOKAHEAD;
02784  6                                   DO WHILE (K := LOOK1(I)) <> 0 AND
02785  6                                                  TOKEN <> K;
02786  6                                        I = I + 1;
02787  7                                        END;
02788  6                                   STATE = LOOK2(I);
02789  6                              END;
02790  5                         ELSE  /* PUSH STATE */
02791  5                              DO;
02792  5                                   CALL INCSP;
02793  6                                   STATESTACK(SP) = GETIN2;
02794  6                                   STATE = GETIN1;
02795  6                              END;
02796  5          END;  /* OF WHILE COMPILING */
02797  4     END;  /* OF WHILE PASS1 OR PASS2) */
02798  3
02799  3     LISTSOURCE = TRUE;
02800  3     CALL INITIALIZE;
02801  3     PASS2 = TRUE;
02802  3     END;  /* OF OF FOREVER  */
02803  2
02804  2     END;   /* OF BLOCK FOR PARSER  */
02805  1
02806  1     EOF
NO PROGRAM ERRORS
```

PROGRAM LISTING - BUILD BASIC-E MACHINE

```
8080 PLM1 VERS 4.1

00001  1    2000H:    /* THIS IS .MEMORY FOR BASIC1 PROGRAM */
00002  1         /*
00003  1              ************************************************
00004  1              *                                              *
00005  1              *              BASIC-E BUILD PROGRAM           *
00006  1              *                                              *
00007  1              *        U. S. NAVY POSTGRADUATE SCHOOL        *
00008  1              *             MONTEREY, CALIFORNIA             *
00009  1              *                                              *
00010  1              *        WRITTEN BY GORDON EUBANKS, JR.        *
00011  1              *                                              *
00012  1              *              CPM VERSION 1.2                 *
00013  1              *                                              *
00014  1              *               DECEMBER 1976                  *
00015  1              *                                              *
00016  1              ************************************************
00017  1         */
00018  1         /*
00019  1              ************************************************
00020  1              *                                              *
00021  1              *      THE BUILD PROGRAM GAINS CONTROL WHEN THE *
00022  1              *      RUN TIME MONITOR IS EXECUTED.  THE INT FILE *
00023  1              *      FOR THE PROGRAM TO BE EXECUTED IS OPENED *
00024  1              *      AND THE BASIC-E MACHINE IS BUILT.        *
00025  1              *                                              *
00026  1              *      BUILD PERFORMS THE FOLLOWING FUNCTIONS:  *
00027  1              *                                              *
00028  1              *      (1) THE NUMERIC CONSTANTS ARE READ FROM  *
00029  1              *      THE INT FILE, CONVERTED TO INTERNAL REP- *
00030  1              *      RESENTATION, AND STORED IN THE FSA.      *
00031  1              *                                              *
00032  1              *      (2) THE SIZE OF THE CODE AREA, DATA AREA *
00033  1              *      AND NUMBER OF PRT ENTRIES ARE READ FROM  *
00034  1              *      THE INT FILE. BUILD THEN DETERMINES THE  *
00035  1              *      ABSOLUTE ADDRESS OF EACH SECTION OF THE  *
00036  1              *      BASIC-E MACHINE. THESE ADDRESSES ARE     *
00037  1              *      PASSED TO THE INTERP PROGRAM VIA FIXED   *
00038  1              *      ADDRESSES IN THE FLOATING POINT SCRATCH  *
00039  1              *      PAD.                                     *
00040  1              *                                              *
00041  1              *      (3) FINALLY INSTRUCTIONS ARE READ FROM   *
00042  1              *      THE FILE AND PLACED IN EITHER THE DATA   *
00043  1              *      AREA OR THE CODE AREA. IN THE CASE OF BRS *
00044  1              *      BRC, PRO, CON, AND DEF OPERATORS THE     *
00045  1              *      ADDRESS FOLLOWING THE INSTRUCTION IS RE- *
00046  1              *      LOCATED TO REFLECT ACTUAL MACHINE ADDRESSES *
00047  1              *      (MINUS 1 BECAUSE PROGRAM COUNTER GETS    *
00048  1              *      INCREMENTED PRIOR TO USE (EXCEPT FOR CON)  ) *
00049  1              *      AFTER (REPEAT AFTER) THE MACHINE HAS BEEN *
00050  1              *      REPOSITIONED BY INTERP. THE END OF THE INT *
00051  1              *      FILE IS INDICATED BY A MACHINE INSTRUCTION *
00052  1              *      7FH.                                     *
00053  1              *      REPOSITIONED BY INTERP.                  *
00054  1              *                                              *
00055  1              ************************************************
00056  1         */
00057  1         /*
00058  1              ************************************************
00059  1              *                                              *
00060  1              *                GLOBAL LITERALS               *
00061  1              *                                              *
00062  1              ************************************************
00063  1         */
00064  1    DECLARE
00065  1              LIT       LITERALLY    'LITERALLY',
00066  1              TRUE      LIT                '1',
00067  1              FALSE     LIT                '0',
00068  1              CR        LIT              '0DH',
00069  1              LF        LIT              '0AH';
00070  1         /*
00071  1              ************************************************
00072  1              *                                              *
00073  1              *          SYSTEM PARAMETERS WHICH MAY         *
00074  1              *          REQUIRE MODIFICATION BY USERS       *
00075  1              *                                              *
00076  1              ************************************************
00077  1         */
00078  1    DECLARE
00079  1              /* OP CODES FOR BASIC-E MACHINE INSTRUCTIONS */
00080  1              DAT       LIT       '51',
00081  1              ILS       LIT       '28',
00082  1              DEF       LIT       '94',
00083  1              BRS       LIT       '54',
00084  1              BRC       LIT       '55',
00085  1              PRO       LIT       '30',
00086  1              CON       LIT       '46';
00087  1         /*
00088  1              ************************************************
00089  1              *                                              *
00090  1              *              EXTERNAL ENTRY POINTS           *
00091  1              *    THESE ENTRY POINTS ALLOW INTERFACEING WITH CP/M *
00092  1              *                                              *
00093  1              ************************************************
00094  1         */
```

```
00095  1    DECLARE
00096  1            BDOS            LIT     '05H', /* ENTRY TO CP/M */
00097  1            BOOT            LIT      '0H', /* RETURN TO SYSTEM */
00098  1            BDOSBEGIN       ADDRESS INITIAL(06H), /* PTR TO BOTTOM CP/M */
00099  1            MAX             BASED   BDOSBEGIN   ADDRESS,
00100  1
00101  1            /* ENTRY POINTS TO OTHER MODULES */
00102  1
00103  1            FPINPUT         LIT    '103H', /* FLT PT INPUT CONVERSION */
00104  1            FPRTN           LIT    '19DH', /* FLT PT OP AND RETURN VALUE */
00105  1            FPNR            LIT    '1A2H', /* FLT PT OP AND NO RETURN */
00106  1            BEGIN           LIT   '2000H', /* TOP OF INTERP - BEGIN BUILD */
00107  1            INTERPENTRY     LIT    '0C00H', /* ENTRY TO INTERPRETER */
00108  1            OFFSET          LIT    '400H', /* SIZE OF BUILD WHICH IS
00109  1                                            AMOUNT TO RELOCATE MACHINE
00110  1                                            ON ENTRY TO INTERP */
00111  1            /* PARAMETER PASSING LOCATIONS */
00112  1
00113  1            PARAM1          LIT   '0BF8H',
00114  1            PARAM2          LIT   '0BFAH',
00115  1            PARAM3          LIT   '0BFCH',
00116  1            PARAM4          LIT   '0BFEH';
00117  1             /*
00118  1             ************************************************************
00119  1             *                                                          *
00120  1             *                    GLOBAL VARIABLES                      *
00121  1             *                                                          *
00122  1             ************************************************************
00123  1            */
00124  1    DECLARE
00125  1            CODEBASE  ADDRESS    INITIAL(PARAM1),
00126  1            DATABASE  ADDRESS    INITIAL(PARAM2),
00127  1            PRTBASE   ADDRESS    INITIAL(PARAM3),
00128  1            STACKBASE ADDRESS    INITIAL(PARAM4),
00129  1            SB        BASED      STACKBASE   ADDRESS, /* FINAL STACK LOC */
00130  1            MPR       BASED      PRTBASE     ADDRESS, /* FINAL PRT LOC */
00131  1            MDA       BASED      DATABASE    ADDRESS, /* FINAL DATA LOC */
00132  1            MCD       BASED      CODEBASE    ADDRESS, /* FINAL CODE LOC */
00133  1            MBASE     ADDRESS, /* PTR TO NEXT POSTION IN DATA AREA */
00134  1            MF        BASED      MBASE       BYTE,
00135  1            BASE      ADDRESS, /* PTR TO NEXT POSITION IN CODE AREA */
00136  1            CURCHAR   BYTE, /* HOLDS CHAR BEING ANALYZED */
00137  1            B         BASED      BASE BYTE,
00138  1            A         BASED      BASE ADDRESS,
00139  1            AP        BYTE, /* ACCUMULATOR INDEX */
00140  1            ACCUM(16) BYTE, /* HOLDS CONSTANTS PRIOR TO CONV */
00141  1            TEMP      ADDRESS,
00142  1            T         BASED      TEMP BYTE;
00143  1        /*
00144  1             ************************************************************
00145  1             *                                                          *
00146  1             *          FLOATING POINT INTERFACE ROUTINES               *
00147  1             *                                                          *
00148  1             ************************************************************
00149  1        */
00150  1
00151  1    FPN: PROCEDURE(FUNCTION,LOCATION);
00152  2        DECLARE
00153  2                FUNCTION  BYTE,
00154  2                LOCATION  ADDRESS;
00155  2        GOTO FPNR;
00156  2    END FPN;
00157  1
00158  1
00159  1    FP: PROCEDURE(FUNCTION,LOCATION);
00160  2        DECLARE
00161  2                FUNCTION BYTE,
00162  2                LOCATION ADDRESS;
00163  2      GO TO FPRTN;
00164  2    END FP;
00165  1
00166  1    FPINP: PROCEDURE(COUNT,LOCATION);
00167  2        DECLARE
00168  2                COUNT    BYTE,
00169  2                LOCATION ADDRESS;
00170  2        GO TO FPINPUT;
00171  2    END FPINP;
00172  1
00173  1        /*
00174  1             ************************************************************
00175  1             *                                                          *
00176  1             *                  CP/M INTERFACE ROUTINES                 *
00177  1             *                                                          *
00178  1             ************************************************************
00179  1        */
00180  1    DECLARE
00181  1            DISKBUFFLOC LIT     '80H',
00182  1            FCBLOC      LIT     '5CH',
00183  1            DISKBUFFEND LIT    '100H',
00184  1          /* IF OPERATING SYSTEM READS VARIABLE LENGTH RECORDS
00185  1             THIS MUST BE ADDRESS OF ACTUAL END OF RECORD */
00186  1            BUFF        ADDRESS   INITIAL(DISKBUFFEND), /* INPUT BUFFER */
00187  1            CHAR        BASED     BUFF        BYTE,     /* INPUTBUFF POINTER */
00188  1            FILENAME    ADDRESS   INITIAL   (FCBLOC),
00189  1            FNP         BASED     FILENAME  BYTE; /* FILE CONTROL BLK */
00190  1
```

```
00191  1
00192  1    MON1:PROCEDURE(FUNCTION,PARAMETER);
00193  2        DECLARE
00194  2                FUNCTION  BYTE,
00195  2                PARAMETER   ADDRESS;
00196  2        GO TO 0005H;
00197  2    END MON1;
00198  1
00199  1    MON2: PROCEDURE(FUNCTION,PARAMETER) BYTE;
00200  2        DECLARE
00201  2                FUNCTION    BYTE,
00202  2                PARAMETER   ADDRESS;
00203  2        GO TO 0005H;
00204  2    END MON2;
00205  1
00206  1
00207  1    MON3: PROCEDURE;
00208  2        HALT; /* FOR OMRON SYSTEMS */
00209  2        GO TO 0C07H;
00210  2    END MON3;
00211  1
00212  1
00213  1    PRINTCHAR: PROCEDURE(CHAR);
00214  2        DECLARE CHAR BYTE;
00215  2        CALL MON1(2,CHAR);
00216  2    END PRINTCHAR;
00217  1
00218  1
00219  1    PRINT: PROCEDURE(BUFFER);
00220  2         /*
00221  2             PRINT A LINE ON CONSOLE FOLLOWED BY A
00222  2             CARRIAGE RETURN AND LINEFEED
00223  2         */
00224  2        DECLARE
00225  2                BUFFER ADDRESS;
00226  2        CALL MON1(9,BUFFER);
00227  2        CALL PRINTCHAR(CR);
00228  2        CALL PRINTCHAR(LF);
00229  2    END PRINT;
00230  1
00231  1
00232  1    OPEN$INT$FILE: PROCEDURE;
00233  2        FNP(9) = 'I';
00234  2        FNP(10) = 'N';
00235  2        FNP(11) = 'T';
00236  2        IF MON2(15,FILENAME) = 255 THEN
00237  2             DO;
00238  2                  CALL PRINT(.'NI    $');
00239  3                  CALL MON3;
00240  3             END;
00241  2    END OPEN$INT$FILE;
00242  1
00243  1
00244  1    READ$INT$FILE: PROCEDURE BYTE;
00245  2         /*
00246  2            NEXT RECORD IS READ FROM INT FILE
00247  2            DISKBUFFEND MUST REFLECT THE ADDRESS
00248  2            OF THE END OF THE RECORD PLUS ONE
00249  2            FOR FIXED SIZE RECORDS THIS IS A CONSTANT
00250  2            RETURNS ZERO IF READ IS SAT, AND 1 IF EOF
00251  2         */
00252  2        RETURN MON2(20,FILENAME);
00253  2    END READ$INT$FILE;
00254  1        /*
00255  1             ***************************************************
00256  1             *                                                 *
00257  1             *                  GLOBAL PROCEDURES              *
00258  1             *                                                 *
00259  1             ***************************************************
00260  1        */
00261  1
00262  1
00263  1    INCBUF:  PROCEDURE;
00264  2        IF(BUFF := BUFF + 1) >= DISKBUFFEND THEN
00265  2             DO;
00266  2                  BUFF = DISKBUFFLOC;
00267  3                  IF READ$INT$FILE <> 0 THEN
00268  3                       CHAR = 7FH;
00269  3             END;
00270  2    END INCBUF;
00271  1
00272  1
00273  1    STC$CHAR$INC: PROCEDURE;
00274  2         /*
00275  2            GET NEXT CHAR FROM INT FILE AND
00276  2            PLACE IN CODE AREA. THEN INCREMENT
00277  2            PTR INTO CODE AREA.
00278  2         */
00279  2        B=CHAR;
00280  2        BASE=BASE+1;
00281  2    END STO$CHAR$INC;
00282  1
00283  1
00284  1    NEXT$CHAR: PROCEDURE BYTE;
00285  2        CALL INCBUF;
00286  2        RETURN CURCHAR := CHAR;
00287  2    END NEXTCHAR;
00288  1
```

```
00289  1
00290  1   GET$TWO$BYTES: PROCEDURE;
00291  2       /*
00292  2         GET NEXT TWO BYTES FROM THE INT FILE
00293  2         AND PLACE THEM IN THE CODE AREA IN REVERSE ORDER.
00294  2       */
00295  2       B(1) = NEXT$CHAR;
00296  2       B = NEXTCHAR;
00297  2       RETURN;
00298  2   END GET$TWO$BYTES;
00299  1
00300  1
00301  1   INC$BASE$TWO: PROCEDURE;
00302  2       BASE = BASE + 1 + 1;
00303  2       RETURN;
00304  2   END INC$BASE$TWO;
00305  1
00306  1
00307  1   GETPARM: PROCEDURE ADDRESS;
00308  2       /*
00309  2          READ A 16 BIT PARAMETER FROM INT FILE
00310  2          AND CONVERT IT TO AN 8080 ADDRESS QUANTITY
00311  2       */
00312  2       RETURN SHL(DOUBLE(NEXTCHAR),8) + NEXTCHAR;
00313  2   END GETPARM;
00314  1
00315  1       /*
00316  1            ************************************************************
00317  1            *                                                          *
00318  1            *                EXECUTION BEGINS HERE                     *
00319  1            *                                                          *
00320  1            ************************************************************
00321  1       */
00322  1   CALL PRINT(.'NBASIC INTERPRETER - VER 1.2$');
00323  1
00324  1   BASE = .MEMORY; /* THIS IS BEGINNING OF MACHINE AND FDA */
00325  1
00326  1   CALL FPN(0,0); /* INITIALIZE FLOATING POINT PACKAGE */
00327  1         /*
00328  1              PROCESS CONSTANTS
00329  1              EACH CONSTANT IS SEPARATED BY A $
00330  1              LAST CONSTANT FOLLOWED BY A *
00331  1         */
00332  1   DO WHILE(ACCUM := NEXT$CHAR) <> '*'; /* * INDICATES END OF CONST */
00333  1       AP = 0;    /* COUNTER FOR LENGTH OF THIS CONSTANT */
00334  2       DO WHILE(ACCUM(AP:=AP+1) := NEXT$CHAR) <> '$';
00335  2             /* GET CONSTANT INTO THE ACCUM */
00336  2             END;
00337  2       CALL FPINP(AP,.ACCUM); /* CONVERT IT TO INTERNAL FORM */
00338  2       CALL FP(9,BASE);  /* LOAD INTO FDA FROM F/P ACCUM */
00339  2       BASE = BASE + 4; /* NEXT LOCATION */
00340  2       END;     /* OF LOOKING FOR *  */
00341  1
00342  1         /*
00343  1              SETUP MACHINE ADDRESS
00344  1              BASE WILL NOW BE NEXT POSITION IN CODE AREA
00345  1              MBASE WILL BE NEXT POSTION IN DATA AREA
00346  1
00347  1              ACTUAL ADDRESSES OF CODE AREA, DATA AREA
00348  1              PRT, AND STACK ARE PASSED TO INTERPRETER
00349  1              USING FIXED LOCATIONS
00350  1         */
00351  1   MBASE = GETPARM + BASE;
00352  1
00353  1   MDA = MBASE - OFFSET; /* ACTUAL DATA AREA ADDR */
00354  1   MCD = BASE - OFFSET; /* ACTUAL CODE AREA ADDR */
00355  1   MPR = GETPARM + MDA;  /* ACTUAL BEGINNING OF PRT */
00356  1   IF MPR >= MAX THEN /* INSURE THERE IS ENOUGH MEMORY */
00357  1       DO;
00358  1             CALL PRINT(.'NM  $');
00359  2             CALL MON3;
00360  2       END;
00361  1   SB = SHL(GETPARM,2) + MPR; /* NUMBER OF ENTRIES IN PRT * 4=SIZE PRT */
00362  1
00363  1         /*
00364  1              BUILD MACHINE - ATLAST
00365  1              AS OPCODES ARE READ THEY MAY BE:
00366  1                 (1) DAT - WHICH MEANS ALL CHARACTERS
00367  1                 FOLLOWING DAT GO INTO DATA AREA UNTIL
00368  1                 A BINARY ZERO IS INCOUNTERED
00369  1
00370  1                 (2) GREATER THAN 127 - WHICH IS A LIT
00371  1                 OR A LIT. TREAT THIS AS 16 BIT OPCODE
00372  1                 AND PUT IN CODE AREA IN ORDER THEY ARE
00373  1                 ON INT FILE
00374  1
00375  1                 (3) ILS - WHICH MEANS ALL CHARACTERS
00376  1                 FOLLOWING GO INTO CODE AREA UNTIL A
00377  1                 BINARY ZERO IS INCOUNTERED - BUT FIRST
00378  1                 PUT A ILS IN CODE AREA AND THE NEXT
00379  1                 BYTE IS SET TO ZERO AND INCREMENTED
00380  1                 FOR EACH CHARACTER IN THE STRING. IE
00381  1                 A STRING CONSTANT IS A ILS OPCODE,
00382  1                 A LENGTH AND THE STRING.
00383  1
```

```
00384  1                  (4) A NORMAL OP CODE - PUT IN CODE
00385  1                  AREA - BUT IF IT IS A BRS OR BRC OR
00386  1                  DEF OR PRO THEN THE NEXT TWO BYTES
00387  1                  ARE AN ADDRESS WHICH MUST BE RELOCATED
00388  1                  TO THE ACTUAL CODE AREA MINUS 1;
00389  1                  OR IT COULD BE A CON WHICH IS
00390  1                  RELOCATED TO THE FDA.
00391  1            */
00392  1
00393  1    DO WHILE NEXT$CHAR <> 7FH; /* BUILD MACHINE */
00394  1        IF CURCHAR = DAT THEN /* PROCESS DATA STATEMENT */
00395  2            DO WHILE(MF := NEXTCHAR) <> 0; /* LOOK FOR END */
00396  2                MBASE = MBASE + 1;
00397  3                END;
00398  2            ELSE
00399  2                IF CURCHAR >= 128 THEN /* PROCESS LIT OR LID */
00400  2                    DO;
00401  2                        CALL STO$CHAR$INC;
00402  3                        CALL INCBUF;
00403  3                        CALL STO$CHAR$INC;
00404  3                    END;
00405  2                ELSE
00406  2                    IF CURCHAR = ILS THEN /* PROCESS INLINE STRING */
00407  2                        DO;
00408  2                            CALL STO$CHAR$INC;
00409  3                            TEMP = BASE;
00410  3                            CHAR = 0; /* TO SET LENGTH TO 0 INITIAL */
00411  3                            CALL STO$CHAR$INC;
00412  3                            DO WHILE NEXTCHAR <> 0;
00413  3                                CALL STO$CHAR$INC;
00414  4                                T = T + 1;
00415  4                                END;
00416  3                        END;
00417  2                    ELSE
00418  2                        DO;
00419  2                            CALL STO$CHAR$INC;
00420  3                            IF (CURCHAR = BRS) OR (CURCHAR = BRC) OR
00421  3                               (CURCHAR = DEF) OR (CURCHAR = PRO) THEN
00422  3                                DO;
00423  3                                    CALL GET$TWO$BYTES;
00424  4                                    A = A + MCD - 1;
00425  4                                    CALL INC$BASE$TWO;
00426  4                                END;
00427  3                            ELSE
00428  3                                IF CURCHAR = CON THEN
00429  3                                    DO;
00430  3                                      CALL GET$TWO$BYTES;
00431  4                                      A = SHL(A,2) + BEGIN;
00432  4                                      CALL INC$BASE$TWO;
00433  4                                    END;
00434  3                        END;
00435  2    END;   /* LOOKING FOR 7FH */
00436  1    GOTO INTERPENTRY;      /* ENTRY POINT TO INTERP */
00437  1    EOF
NO PROGRAM ERRORS
```

```
8080 PLM1 VERS 4.1

00001  1
00002  1    0C00H:  /*LOAD POINT ABOVE FP PACKAGE */
00003  1        /*
00004  1          ***************************************************
00005  1          *                                                 *
00006  1          *                  BASIC-E INTERPRETER            *
00007  1          *                                                 *
00008  1          *            U. S. NAVY POSTGRADUATE SCHOOL       *
00009  1          *                 MONTEREY, CALIFORNIA            *
00010  1          *                                                 *
00011  1          *            WRITTEN BY GORDON EUBANKS, JR.       *
00012  1          *                                                 *
00013  1          *                  CPM VERSION 1.2                *
00014  1          *                  NOVEMBER 1976                  *
00015  1          *                                                 *
00016  1          ***************************************************
00017  1        */
00018  1
00019  1        /*
00020  1          ***************************************************
00021  1          *                                                 *
00022  1          *     THE BASIC-E INTERPRETER IS PASSED CONTROL    *
00023  1          *      FROM THE BUILD PROGRAM.  THE FDA, CODE AND  *
00024  1          *      DATA AREA ARE MOVED DOWN TO RESIDE AT THE   *
00025  1          *      .MEMORY FOR THIS PROGRAM, AND THEN THE STACK*
00026  1          *      PRT AND MACHINE REGISTERS ARE INITIALIZED   *
00027  1          *      THE INTERPRETER THEN EXECUTES THE BASIC-E    *
00028  1          *      MACHINE CODE.                               *
00029  1          *                                                 *
00030  1          ***************************************************
00031  1        */
00032  1
00033  1        /*
00034  1          ***************************************************
00035  1          *                                                 *
00036  1          *                  GLOBAL LITERALS                *
00037  1          *                                                 *
00038  1          ***************************************************
00039  1        */
00040  1    DECLARE
00041  1              LIT       LITERALLY 'LITERALLY',
00042  1              ADDR      LIT       'ADDRESS',
00043  1              FOREVER   LIT       'WHILE TRUE',
00044  1              TRUE      LIT       '1',
00045  1              FALSE     LIT       '0',
00046  1              LF        LIT       '10',
00047  1              CR        LIT       '13',
00048  1              CONTZ     LIT       '1AH',
00049  1              QUOTE     LIT       '22H',
00050  1              WHAT      LIT       '63';              /*QUESTION MARK*/
00051  1
00052  1        /*
00053  1          ***************************************************
00054  1          *                                                 *
00055  1          *              EXTERNAL ENTRY POINTS              *
00056  1          *     THESE ENTRY POINTS ASSUME THE USE OF CP/M    *
00057  1          *                                                 *
00058  1          ***************************************************
00059  1        */
00060  1    DECLARE
00061  1              BOOT          LIT         '0H',  /*  TO RETURN TO SYSTEM */
00062  1              BDOS          LIT         '5H',  /* ENTRY POINT TO CP/M */
00063  1              SYSBEGIN      ADDRESS  INITIAL(6H),
00064  1              OVERFLOW      ADDRESS  INITIAL(0B2EH),
00065  1              CONBIN        LIT       '157H',  /* CONV TO BINARY ENTRY */
00066  1              CONFP         LIT       '168H',  /* CONV TO FLOAT PT ENTRY */
00067  1              FPINT         LIT       '103H',  /* CONV ASCII TO FLOAT PT */
00068  1              FPOT          LIT       '114H',  /* CONV FLOAT PT TO ASCII */
00069  1              FPRTN         LIT       '190H',  /* OPERATION AND RETURN VALUE */
00070  1              FPNR          LIT       '1A2H',  /* OPERATION NO RETURN VALUE */
00071  1              SEEDLOCATION  LIT      '0B53H',  /* RANDOM NUMBER SEED LOC */
00072  1              BUILDTCP      LIT      '3100H',  /* .MEMORY BUILD PROGRAM */
00073  1              MOVEENTRY     LIT      '0AA5H',  /* MOVE ROUTINE ENTRY */
00074  1              MOVE4ENTRY    LIT      '0AC2H',  /* 4 BYTE MOVE ROUTINE */
00075  1              PORTIN        LIT      '0AE0H',  /* PORT INPUT ROUTINE */
00076  1              PORTOUT       LIT      '0AF0H',  /* PORT OUTPUT ROUTINE */
00077  1              RANDOMLOC     LIT      '0A5BH';  /* RANDOM NUMBER GENERATOR */
00078  1
00079  1        /*
00080  1          ***************************************************
00081  1          *                                                 *
00082  1          *            SYSTEM PARAMETERS WHICH MAY          *
00083  1          *            REQUIRE MODIFICATION BY USERS        *
00084  1          *                                                 *
00085  1          ***************************************************
00086  1        */
00087  1    DECLARE
00088  1              EOLCHAR       LIT       '0DH',
00089  1              EOFFILLER     LIT       '1AH',
00090  1              INTRECSIZE    LIT       '128',
00091  1              STRINGDELIM   LIT       '22H',
00092  1              CONBUFFSIZE   LIT        '80',
00093  1              NUMFILES      LIT         '6',  /* MAX NUMBER USER FILES */
00094  1              NRSTACK       LIT        '40';  /* STACK SIZE TIMES 4 */
00095  1
```

```
00096  1        /*
00097  1             ************************************************************
00098  1             *                                                          *
00099  1             *                      GLOBAL VARIABLES                    *
00100  1             *                                                          *
00101  1             ************************************************************
00102  1        */
00103  1
00104  1    DECLARE
00105  1             RA          ADDRESS,  /* ADDRESS OF REG A */
00106  1             RB          ADDRESS,  /* ADDRESS OF REG B */
00107  1             RC          ADDRESS,  /* ADDRESS OF REGISTER C */
00108  1             C           BASED     RC BYTE, /* BYTE OF CODE */
00109  1             TWOBYTEOPRAND BASED RC ADDRESS,  /* TWO BYTES CODE */
00110  1             SB          ADDRESS,  /* BOTTOM OF STACK */
00111  1             ST          ADDRESS, /* TOP OF STACK */
00112  1             BRA         BASED     RA          BYTE,
00113  1             ARA         BASED     RA          ADDRESS,
00114  1             ARB         BASED     RB          ADDRESS,
00115  1             BRB         BASED     RB          BYTE,
00116  1             MPR         ADDRESS,  /* BASE ADDRESS OF PRT */
00117  1             MDA         ADDRESS,  /* BASE OF DATA AREA */
00118  1             MCD         ADDRESS,  /* BASE OF CODE AREA */
00119  1             CURRENTLINE ADDRESS INITIAL(0),  /* SOURCE LIME BEING EXEC */
00120  1             DATAAREAPTR ADDRESS,  /* CURRENT LOCATION IN DATA AREA */
00121  1             SEEDLOC ADDRESS INITIAL(SEEDLOCATION),
00122  1             MBASE       ADDRESS;  /* BEGINNING OF FREE STORAGE AREA */
00123  1
00124  1    DECLARE
00125  1             INPUTBUFFER         BYTE INITIAL(CONBUFFSIZE), /* USED WITH SPACE */
00126  1             SPACE(CONBUFFSIZE) BYTE,  /* INPUT BUFFER FOR CON AND DISK */
00127  1             INPUTINDEX          BYTE,
00128  1             CONBUFFPTR          ADDRESS,
00129  1             INPUTPTR            ADDRESS,
00130  1             PRINTBUFFLENGTH     LIT       '71',
00131  1             PRINTBUFFERLOC      LIT       '80H',
00132  1             TABPOS1             LIT       '142', /* ABSOLUTE ADDR REL TO */
00133  1             TABPOS2             LIT       '156', /*   PRINTBUFFLOC  */
00134  1             TABPOS3             LIT       '170',
00135  1             TABPOS4             LIT       '184',
00136  1             PRINTBUFFER         ADDRESS   INITIAL(PRINTBUFFERLOC),
00137  1             PRINTPCS            BASED     PRINTBUFFER   BYTE,
00138  1             PRINTBUFFEND        LIT       '0C7H', /* ABSOLUTE ADDRESS */
00139  1             PRINTWORKAREA(14)  BYTE,  /* FOR CONV FROM FP TO ASCII */
00140  1             REREADADDR          ADDRESS,  /* TO RECOVER FROM READ ERROR */
00141  1             INPUTTYPE           BYTE;
00142  1
00143  1    DECLARE
00144  1             FILEADDR              ADDRESS, /*CURRENT FCB POINTER BASE */
00145  1             FCB                   BASED     FILEADDR  BYTE,
00146  1             FCBADD                BASED     FILEADDR  ADDRESS,
00147  1             EOFADDR               ADDRESS,
00148  1             FILES(NUMFILES)       ADDRESS,  /*POINTER ARRAY TO FCBS */
00149  1             EOFBRANCH(NUMFILES)  ADDRESS,
00150  1             BUFFER$END            ADDRESS,
00151  1             RECORD$POINTER        ADDRESS,
00152  1             BUFFER                ADDRESS,
00153  1             NEXTDISKCHAR          BASED     RECORD$POINTER BYTE,
00154  1             BLOCKSIZE             ADDRESS,
00155  1             BYTES$WRITTEN         ADDRESS,
00156  1             FIRSTFIELD            BYTE,
00157  1             EOFRA                 ADDRESS,
00158  1             EOFRB                 ADDRESS;
00159  1
00160  1
00161  1        /*
00162  1             ************************************************************
00163  1             *                                                          *
00164  1             *       SYSTEM DEPENDENT ROUTINES AND VARIABLES            *
00165  1             *           THE FOLLOWING ROUTINES ARE USED                *
00166  1             *           BY THE INTERPRETER TO ACCESS DISK              *
00167  1             *           FILES AND FOR CONSOLE I/O.                     *
00168  1             *           THE ROUTINES ASSUME THE USE OF THE             *
00169  1             *           CP/M OPERATING SYSTEM.                         *
00170  1             *                                                          *
00171  1             ************************************************************
00172  1        */
00173  1
00174  1
00175  1    MON1: PROCEDURE(F,A);
00176  2        DECLARE F BYTE,
00177  2        A ADDRESS;
00178  2        GO TO BDOS;
00179  2    END MON1;
00180  1
00181  1
00182  1    MON2: PROCEDURE(F,A) BYTE;
00183  2        DECLARE F BYTE,
00184  2        A ADDRESS;
00185  2        GO TO BDOS;
00186  2    END MON2;
00187  1
00188  1
00189  1    MON3: PROCEDURE;
00190  2        HALT;  /* FOR OMRON SYSTEMS */
00191  2        GOTO BOOT;  /* FOR THE REST OF THE WORLD */
00192  2    END MON3;
```

```
00193  1
00194  1
00195  1    PRINTCHAR: PROCEDURE(CHAR);
00196  2        DECLARE CHAR BYTE;
00197  2        CALL MON1(2,CHAR);
00198  2    END PRINTCHAR;
00199  1
00200  1
00201  1    CRLF: PROCEDURE;
00202  2        CALL PRINTCHAR(CR);
00203  2        CALL PRINTCHAR(LF);
00204  2    END CRLF;
00205  1
00206  1
00207  1    READCHAR: PROCEDURE BYTE;
00208  2            RETURN MON2(1,0);
00209  2    END READCHAR;
00210  1
00211  1
00212  1    READ: PROCEDURE(A);
00213  2        DECLARE A ADDRESS;
00214  2        /* READ INTO BUFFER AT A+2 */
00215  2        CALL MON1(10,A);
00216  2    END READ;
00217  1
00218  1
00219  1    OPEN: PROCEDURE BYTE;
00220  2        RETURN MON2(15,FILEADDR);
00221  2    END OPEN;
00222  1
00223  1
00224  1    CLOSE: PROCEDURE BYTE;
00225  2        RETURN MON2(16,FILEADDR);
00226  2    END CLOSE;
00227  1
00228  1
00229  1    DISKREAD: PROCEDURE BYTE;
00230  2        RETURN MON2(20,FILEADDR);
00231  2    END DISKREAD;
00232  1
00233  1
00234  1    DISKWRITE: PROCEDURE BYTE;
00235  2        RETURN MON2(21,FILEADDR);
00236  2    END DISKWRITE;
00237  1
00238  1
00239  1    MAKE: PROCEDURE BYTE;
00240  2        CALL MON1(19,FILEADDR);
00241  2        RETURN MON2(22,FILEADDR);
00242  2    END MAKE;
00243  1
00244  1
00245  1    SETDMA: PROCEDURE;  /* SET DMA ADDRESS FOR DISK I/O */
00246  2        CALL MON1(26,BUFFER);
00247  2    END SETDMA;
00248  1
00249  1
00250  1    PRINT: PROCEDURE(A);
00251  2        DECLARE A ADDRESS;
00252  2        /* PRINT THE STRING STARTING AT ADDRESS A UNTIL THE
00253  2        NEXT DOLLAR SIGN IS ENCOUNTERED */
00254  2        CALL MON1(9,A);
00255  2    END PRINT;
00256  1
00257  1
00258  1        /*
00259  1              ***********************************************
00260  1              *                                             *
00261  1              *        GENERAL PURPOSE INTERPRETER ROUTINES *
00262  1              *                                             *
00263  1              ***********************************************
00264  1        */
00265  1    TIMES4: PROCEDURE(N) ADDRESS;
00266  2        DECLARE N ADDRESS;
00267  2        RETURN SHL(N,2);
00268  2    END TIMES4;
00269  1
00270  1
00271  1    PRINT$DEC: PROCEDURE(VALUE);
00272  2        DECLARE VALUE ADDRESS,
00273  2                I BYTE,
00274  2                COUNT BYTE,
00275  2                DECIMAL(4) ADDRESS INITIAL(1000,100,10,1);
00276  2        DO I = 0 TO 3;
00277  2            COUNT = 30H;
00278  3            DO WHILE VALUE >= DECIMAL(I);
00279  3                VALUE = VALUE - DECIMAL(I);
00280  4                COUNT = COUNT + 1;
00281  4                END;
00282  3            CALL PRINTCHAR(COUNT);
00283  3        END;
00284  2    END PRINT$DEC;
00285  1
00286  1
00287  1    MOVE: PROCEDURE(SOURCE,DEST,N);
00288  2
```

```
00289  2        /*MOVE N BYTES FROM SOURCE TO DEST */
00290  2        DECLARE (SOURCE,DEST,N) ADDRESS;
00291  2
00292  2        MOVEA: PROCEDURE(A);
00293  3             DECLARE A ADDRESS;
00294  3             GOTO MOVEENTRY;
00295  3        END MOVEA;
00296  2
00297  2        CALL MOVEA(.SOURCE);
00298  2    END MOVE;
00299  1
00300  1
00301  1    MOVE4: PROCEDURE(SOURCE,DEST);
00302  2        DECLARE SOURCE ADDRESS,
00303  2                DEST ADDRESS;
00304  2        GOTO MOVE4ENTRY;
00305  2    END MOVE4;
00306  1
00307  1
00308  1    FILL: PROCEDURE(DEST,CHAR,N);
00309  2        /*FILL LOCATIONS STARTING AT DEST WITH CHAR FOR N BYTES */
00310  2        DECLARE
00311  2                DEST   ADDRESS,
00312  2                N      ADDRESS,
00313  2                D      BASED    DEST   BYTE,
00314  2                CHAR   BYTE;
00315  2        DO WHILE (N:=N-1) <> OFFFFH;
00316  2             D = CHAR;
00317  3             DEST = DEST + 1;
00318  3        END;
00319  2    END FILL;
00320  1
00321  1
00322  1
00323  1    OUTPUT$MSG: PROCEDURE(MSG);
00324  2        DECLARE MSG ADDRESS;
00325  2        CALL PRINT$CHAR(HIGH(MSG));
00326  2        CALL PRINT$CHAR(LOW(MSG));
00327  2        IF CURRENTLINE > 0 THEN
00328  2             DO;
00329  2                  CALL PRINT(.' IN LINE $');
00330  3             CALL PRINT$DEC(CURRENTLINE);
00331  3             END;
00332  2        CALL CRLF;
00333  2    END OUTPUT$MSG;
00334  1
00335  1
00336  1    ERROR: PROCEDURE(E);
00337  2        DECLARE E ADDRESS;
00338  2        CALL CRLF;
00339  2        CALL PRINT(.'ERROR $');
00340  2        CALL OUTPUT$MSG(E);
00341  2        CALL MON3;
00342  2    END ERROR;
00343  1
00344  1
00345  1    WARNING: PROCEDURE(W);
00346  2        DECLARE W ADDRESS;
00347  2        CALL CRLF;
00348  2        CALL PRINT(.'WARNING $');
00349  2        CALL OUTPUT$MSG(W);
00350  2        RETURN;
00351  2    END WARNING;
00352  1
00353  1
00354  1        /*
00355  1             ************************************************
00356  1             *                                              *
00357  1             *          STACK MANIPULATION ROUTINES         *
00358  1             *                                              *
00359  1             ************************************************
00360  1        */
00361  1
00362  1    STEP$INS$CNT: PROCEDURE;
00363  2        RC=RC+1;
00364  2    END STEP$INS$CNT;
00365  1
00366  1    POP$STACK: PROCEDURE;
00367  2        RA = RB;
00368  2        IF(RB := RB - 4) < SB THEN
00369  2             RB = ST - 4;
00370  2    END POP$STACK;
00371  1
00372  1    PUSH$STACK: PROCEDURE;
00373  2        RB = RA;
00374  2        IF(RA := RA + 4) >= ST THEN
00375  2             RA = SB;
00376  2    END PUSH$STACK;
00377  1
00378  1
00379  1    IN$FSA: PROCEDURE(A) BYTE;
00380  2        /*
00381  2             RETURNS TRUE IF A IS IN FSA
00382  2        */
00383  2        DECLARE A ADDRESS;
00384  2        RETURN A > ST;
00385  2    END IN$FSA;
00386  1
```

```
00387  1
00388  1    SET$DATA$ADDR: PROCEDURE(PTR);
00389  2        DECLARE PTR ADDRESS, A BASED PTR ADDRESS;
00390  2        IF NOT INSFSA(A) THEN
00391  2            A = MPR + TIMES4(A);
00392  2    END SET$DATA$ADDR;
00393  1
00394  1
00395  1    MOVE$RA$RB: PROCEDURE;
00396  2        CALL MOVE4(RA,RB);
00397  2    END MOVE$RA$RB;
00398  1
00399  1
00400  1    MOVE$RB$RA:  PROCEDURE;
00401  2        CALL MOVE4(RB,RA);
00402  2            END MOVERBRA;
00403  1
00404  1
00405  1    FLIP: PROCEDURE;
00406  2        DECLARE TEMP(4) BYTE;
00407  2        CALL MOVE4(RA,.TEMP);
00408  2        CALL MOVE$RB$RA;
00409  2        CALL MOVE4(.TEMP,RB);
00410  2            END FLIP;
00411  1
00412  1
00413  1    LOAD$RA: PROCEDURE;
00414  2        CALL SET$DATA$ADDR(RA);
00415  2        CALL MOVE4(ARA,RA);
00416  2        END LOADRA;
00417  1
00418  1    RA$ZERO: PROCEDURE BYTE;
00419  2        RETURN BRA = 0;
00420  2    END RA$ZERO;
00421  1
00422  1
00423  1    RB$ZERO: PROCEDURE BYTE;
00424  2        RETURN BRB = 0;
00425  2    END RB$ZERO;
00426  1
00427  1
00428  1    RA$ZERO$ADDRESS: PROCEDURE BYTE;
00429  2        RETURN ARA = 0;
00430  2    END RA$ZERO$ADDRESS;
00431  1
00432  1
00433  1    RB$ZERO$ADDRESS: PROCEDURE BYTE;
00434  2        RETURN ARB = 0;
00435  2    END RB$ZERO$ADDRESS;
00436  1
00437  1
00438  1    RA$NEGATIVE: PROCEDURE BYTE;
00439  2        RETURN RCL(BRA(1),1);
00440  2    END RA$NEGATIVE;
00441  1
00442  1
00443  1    RB$NEGATIVE: PROCEDURE BYTE;
00444  2        RETURN RCL(BRB(1),1);
00445  2    END RB$NEGATIVE;
00446  1
00447  1
00448  1    FLAG$STRING$ADDR: PROCEDURE(X);
00449  2        DECLARE X BYTE;
00450  2        BRA(2) = X;
00451  2    END FLAG$STRING$ADDR;
00452  1
00453  1
00454  1        /*
00455  1             ************************************************************
00456  1             *                                                          *
00457  1             *            FLOATING POINT INTERFACE ROUTINES             *
00458  1             *                                                          *
00459  1             *        ALL FLOATING POINT OPERATIONS ARE PERFORMED       *
00460  1             *        BY CALLING ROUTINES IN THIS SECTION.  THE         *
00461  1             *        FLOATING POINT PACKAGE IS ACCESSED BY THE         *
00462  1             *        FOLLOWING SIX ROUTINES:                           *
00463  1             *            (1)  CONV$TO$BINARY                           *
00464  1             *            (2)  CONV$TO$FP                               *
00465  1             *            (3)  FP$INPUT                                 *
00466  1             *            (4)  FP$OUT                                   *
00467  1             *            (5)  FP$OP$RETURN                             *
00468  1             *            (6)  FP$OP                                    *
00469  1             *        CHECK$OVERFLOW DOES JUST THAT                     *
00470  1             *        THE REMAINING ROUTINES USE THE ABOVE              *
00471  1             *        PROCEDURES TO ACCOMPLISH COMMON ROUTINES          *
00472  1             *                                                          *
00473  1             *        CONV$TO$BIN$ADDR AND OTHER ROUTINES WHICH         *
00474  1             *        REFER TO AN ADDRESS PLACE THE RESULTS IN          *
00475  1             *        THE FIRST TWO BYTES OF THE STACK AS AN 8080       *
00476  1             *        ADDRESS QUANTITY WITH LOW ORDER BYTE FIRST        *
00477  1             *                                                          *
00478  1             *                                                          *
00479  1             *        ALL INTERFACING IS DONE USING ABSOLUTE ADDR.      *
00480  1             *                                                          *
00481  1             ************************************************************
00482  1        */
00483  1
```

```
00484  1    DECLARE
00485  1              FINIT      LIT        '0',                  /* INITIALIZE*/
00486  1              FSTR       LIT        '1',                  /* STORE (ACCUM)*/
00487  1              FLOD       LIT        '2',                  /* LOAD ACCUM */
00488  1              FADD       LIT        '3',                  /* ADD TO ACCUM */
00489  1              FSUB       LIT        '4',                  /* SUB FROM ACCUM*/
00490  1              FMUL       LIT        '5',                  /* MUL BY ACCUM*/
00491  1              FDIV       LIT        '6',        /* DIVIDE INTO ACCUM*/
00492  1              FABS       LIT        '7',                  /* ABS VALUE OF ACCUM*/
00493  1              FZRO       LIT        '8',                  /* ZERO ACCUM*/
00494  1              FTST       LIT        '9',                  /* TEST SIGN OF ACCUM*/
00495  1              FCHS       LIT        '10',                 /* COMPL. ACCUM*/
00496  1              SQRT       LIT        '11',                 /* SQRT OF ACCUM*/
00497  1              COS        LIT        '12',                 /* COS ACCUM*/
00498  1              SIN        LIT        '13',                 /* SIN ACCUM*/
00499  1              ATAN       LIT        '14',                 /* ARCTAN ACCUM */
00500  1              COSH       LIT        '15',                 /* COSH ACCUM*/
00501  1              SINH       LIT        '16',                 /* SINH ACCUM*/
00502  1              EXP        LIT        '17',                 /* EXPONENTIAL ACCUM*/
00503  1              LOG        LIT        '18';                 /* LOG ACCUM*/
00504  1
00505  1
00506  1    CHECK$OVERFLOW: PROCEDURE;
00507  2        DECLARE
00508  2                  B       BASED OVERFLOW BYTE,
00509  2                  MAXNUM DATA(0FFH,07FH,0FFH,0FFH);
00510  2        IF B THEN
00511  2              DO;
00512  2                   CALL WARNING('OF');
00513  3                   CALL MOVE4(.MAXNUM,RA);
00514  3                   B = 0;
00515  3              END;
00516  2    END CHECK$OVERFLOW;
00517  1
00518  1
00519  1    CONV$TO$BINARY: PROCEDURE(A);  /*CONVERTS FP NUM AT A TO BINARY
00520  2              AND RETURNS RESULT TO A  */
00521  2        DECLARE A ADDRESS;
00522  2              GOTO CONBIN;
00523  2    END CONV$TO$BINARY;
00524  1
00525  1    CONV$TO$FP: PROCEDURE(A);  /* CONVERTS BINARY NUM AT A TO FP AND
00526  2              LEAVES IT AT A  */
00527  2        DECLARE A ADDRESS;
00528  2              GOTO CONFP;
00529  2    END CONV$TO$FP;
00530  1
00531  1    FP$INPUT: PROCEDURE(LENGTH,A);  /* CONVERTS STRING AT A LENGTH LENGTH
00532  2               TO FP AND LEAVES RESULT IN FP ACCUM    */
00533  2        DECLARE LENGTH BYTE, A ADDRESS;
00534  2              GOTO FPINT;
00535  2    END FP$INPUT;
00536  1
00537  1
00538  1    FP$OUT: PROCEDURE(A);  /* CONVERTS FP ACCUM TO STRING AND PUTS IT
00539  2              AT A  */
00540  2        DECLARE A ADDRESS;
00541  2              GOTO FPOT;
00542  2    END FP$OUT;
00543  1
00544  1
00545  1    FP$OP$RETURN: PROCEDURE(FUNC,A); /* PERFORMS FUNC AND RETURNS VALUE
00546  2              TO A  */
00547  2    DECLARE FUNC BYTE, A ADDRESS;
00548  2              GOTO FPRTN;
00549  2    END FP$OP$RETURN;
00550  1
00551  1
00552  1    FP$OP: PROCEDURE(FUNC,A);  /* PERFORMS FUNC POSSIBLY USEING
00553  2              FP NUM ADDRESSED BY A . NOTHING IS RETURNED TO A */
00554  2        DECLARE FUNC BYTE, A ADDRESS;
00555  2              GOTO FPNR;
00556  2    END FP$OP;
00557  1
00558  1    CONV$TO$BIN$ACCR: PROCEDURE;
00559  2        CALL CONV$TO$BINARY(RA);
00560  2        BRA = BRA(3);
00561  2        BRA(1) = BRA(2);
00562  2    END CONV$TO$BIN$ACCR;
00563  1
00564  1
00565  1    ONE$VALUE$OPS: PROCEDURE(A);
00566  2              DECLARE A BYTE;
00567  2              CALL FP$OP(FLOD,RA);
00568  2              CALL FP$OP$RETURN(A,RA);
00569  2        CALL CHECK$OVERFLOW;
00570  2    END ONE$VALUE$OPS;
00571  1
00572  1    TWO$VALUE$OPS: PROCEDURE(TYPE);
00573  2        DECLARE TYPE BYTE;
00574  2              CALL FP$OP(FLOD,RA);
00575  2              CALL FP$OP$RETURN(TYPE,RB);
00576  2        CALL POP$STACK;
00577  2        CALL CHECK$OVERFLOW;
00578  2    END TWO$VALUE$OPS;
00579  1
```

```
C0580  1    ROUND$CONV$@IN: PROCEDURE;
00581  2        DECLARE CNEHALF DATA(80H,0,0,0);
00582  2        CALL PUSH$STACK;
00583  2        CALL MOVE4(.CNEHALF,RA);
00584  2        CALL TWO$VALUE$OPS(FADD);
00585  2        CALL CONV$TO$BIN$ADDR;
00586  2    END ROUND$CONV$@IN;
00587  1
00588  1    FLOAT$ADDR: PROCEDURE(V);
00589  2        DECLARE V ADDRESS;
C0590  2        RA=0;
00591  2        BRA(2)=HIGH(V); BRA(3)=LOW(V);
00592  2        CALL CONV$TO$FP(RA);
00593  2    END FLOAT$ADDR;
00594  1
00595  1    COMPARE$FP: PROCEDURE BYTE;
00596  2       /* 1=LESS 2=GREATER 3=EQUAL  */
00597  2            CALL FP$OP(FLOD,RB);
00598  2            CALL FP$OP$RETURN(FSUB,RA);
C0599  2            IF RA$ZERO THEN
C0600  2                RETURN 3;
C0601  2            IF RA$NEGATIVE THEN
00602  2                RETURN 1;
00603  2            RETURN 2;
00604  2    END COMPARE$FP;
00605  1
C0606  1
00607  1        /*
00608  1              ***************************************************
00609  1              *                                                 *
00610  1              *          DYNAMIC STORAGE ALLOCATION PROCEDURES  *
00611  1              *                                                 *
00612  1              ***************************************************
00613  1        */
00614  1    AVAILABLE: PROCEDURE(NBYTES) ADDRESS;
00615  2        DECLARE
00616  2                NBYTES    ADDRESS,
C0617  2                POINT     ADDRESS,
00618  2                TEMP      ADDRESS,
00619  2                TOTAL     ADDRESS,
C0620  2                HERE      BASED POINT   ADDRESS,
00621  2                SWITCH    BASED POINT   BYTE;
00622  2        POINT = MBASE;
00623  2        TOTAL = 0;
00624  2        DO WHILE PCINT <> 0;
00625  2            IF SWITCH(4) = 0 THEN
00626  3              DO;
C0627  3                 TOTAL = TOTAL + (TEMP := HERE - POINT - 5);
00628  4                 IF NBYTES <> 0 THEN
C0629  4                   DO;
C0630  4                      IF NBYTES + 5 <= TEMP THEN
00631  5                        RETURN POINT;
00632  5                   END;
00633  4               END;
00634  3               POINT = HERE;
00635  3        END;
00636  2        IF NBYTES <> 0 THEN
C0637  2            CALL ERROR('CM');
00638  2        RETURN TOTAL;
00639  2    END AVAILABLE;
C0640  1
00641  1    GETSPACE: PROCEDURE(NBYTES) ADDRESS;
00642  2        DECLARE
00643  2                NBYTES  ADDRESS,
00644  2                SPACE   ADDRESS,
C0645  2                PCINT   ADDRESS,
00646  2                HERE    BASED POINT   ADDRESS,
C0647  2                TEMP    ADDRESS,
00648  2                TEMP1   ADDRESS,
00649  2                TEMP2   ADDRESS,
C0650  2                ADR1    BASED TEMP1   ADDRESS,
00651  2                ADR2    BASED TEMP2   ADDRESS,
00652  2                SWITCH  BASED POINT   BYTE,
00653  2                SWITCH2 BASED TEMP1   BYTE;
00654  2        IF NBYTES = 0 THEN
C0655  2            RETURN 0;
C0656  2        PCINT = AVAILABLE(NBYTES);
00657  2        /*LINK UP THE SPACE*/
00658  2        SWITCH(4)=1;  /* SET SWITCH ON*/
00659  2            TEMP1=PCINT+NBYTES+5;
C0660  2            ADR1=HERE;
00661  2            TEMP2=HERE + 2;
00662  2           HERE,ADR2 = TEMP1;
00663  2           SWITCH2(4)=0; /*SET REMAINDER AS AVAIL*/
00664  2          TEMP1 = TEMP1 + 2;
00665  2          ADR1 = PCINT;
00666  2        CALL FILL(PCINT := POINT + 5,0,NBYTES);
C0667  2        RETURN PCINT;
00668  2    END GETSPACE;
00669  1
C0670  1    RELEASE: PROCEDURE(SPACE);
00671  2        DECLARE
00672  2                SPACE       ADDRESS,
00673  2                HOLD        ADDRESS,
00674  2                NEXT$AREA   BASED     HOLD    ADDRESS,
00675  2                SWITCH      BASED     SPACE   BYTE,
00676  2                HERE        BASED     SPACE   ADDRESS,
00677  2                TEMP        ADDRESS,
00678  2                ADRS        BASED     TEMP    ADDRESS,
```

```
00679  2                LOCK       BASED     TEMP    BYTE;
00680  2
00681  2        UNLINK: PROCEDURE;
00682  3            TEMP=HERE;
00683  3            IF ADRS<>0 THEN       /*NOT AT TOP OF FSA */
00684  3             DO;
00685  3                IF LOCK(4)=0 THEN    /*SPACE ABOVE IS FREE*/
00686  4                  DO;
00687  4                       TEMP=(HERE:=ADRS) + 2;
00688  5                       ADRS=SPACE;
00689  5                  END;
00690  4             END;
00691  3        END UNLINK;
00692  2
00693  2        HOLD,SPACE=SPACE-5;
00694  2        SWITCH(4)=0;     /* RELEASES THE SPACE */
00695  2        /* COMBINE WITH SPACE ABOVE AND BELOW IF POSSIBLE*/
00696  2        CALL UNLINK;
00697  2        SPACE=SPACE+2;  /* LOOK AT PREVIOUS BLOCK*/
00698  2        IF (SPACE:=HERE)<>0 THEN
00699  2        DO;
00700  2            IF SWITCH(4)=0 THEN
00701  3            DO;
00702  3                CALL UNLINK;
00703  4               HOLD=SPACE;
00704  4            END;
00705  3        END;
00706  2    END RELEASE;
00707  1
00708  1        /*
00709  1             ***************************************************
00710  1             *                                                 *
00711  1             *            ARRAY ADDRESSING PROCEDURES          *
00712  1             *                                                 *
00713  1             *      CALC$ROW SETS UP AN ARRAY IN THE FSA IN ROW *
00714  1             *      MAJOR ORDER.  THE BYTE OF CODE FOLLOWING THE *
00715  1             *      OPERATOR IS THE NUMBER OF DIMENSIONS.  THE  *
00716  1             *      STACK CONTAINS THE UPPER BOUND OF EACH DIMENSION *
00717  1             *      RA HOLDS DIMENSION N, RB DIMENSION N-1 ETC. *
00718  1             *      THE LOWER BOUND IS ALWAYS ZERO.             *
00719  1             *                                                  *
00720  1             *      CALC$SUB PERFORMS A SUBSCRIPT CALCULATION FOR *
00721  1             *      THE ARRAY REFERENCED BY RA.  THE VALUE OF EACH *
00722  1             *      DIMENSION IS ON THE STACK BELOW THE ARRAY    *
00723  1             *      ADDRESS STARTING WITH THE NTH DIMENSION      *
00724  1             *      A CHECK IS MADE TO SEE IF THE SELECTED ELEMENT *
00725  1             *      IS OUTSIDE THE AREA ASIGNED TO THE ARRAY     *
00726  1             *                                                   *
00727  1             ***************************************************
00728  1        */
00729  1
00730  1    CALC$ROW: PROCEDURE;
00731  2        DECLARE
00732  2                ASIZE        ADDRESS,
00733  2                I            BYTE,
00734  2                SAVERA       ADDRESS,
00735  2                SAVERB       ADDRESS,
00736  2                ARRAYADDR    ADDRESS,
00737  2                NUMDIM       BASED RC BYTE,
00738  2                ARRAYPOS     BASED ARRAYADDR ADDRESS;
00739  2
00740  2        ASIZE = 1;  /* INITAIL VALUE */
00741  2        CALL STEP$INS$CNT;  /* POINT RC TO NUMDIM */
00742  2        SAVERA = RA;  /* SAVE CURRENT STACK POINTER */
00743  2        SAVERB = RB;
00744  2        DO I = 1 TO NUMDIM; /* FIRST PASS ON ARRAY DIMENSIONS */
00745  2             ARA,ASIZE = ASIZE * (ARA + 1); /* DISPLACEMENT AND TOTAL */
00746  3             CALL POP$STACK;  /* NEXT DIMENSION */
00747  3             END;
00748  2        RA = SAVERA;  /* BACK TO ORIGINAL STACK POSITION */
00749  2        RB = SAVERB;
00750  2        SAVERA,ARRAYADDR = GETSPACE(TIMES4(ASIZE) + SHL(NUMDIM+1,1));
00751  2        ARRAYPOS = NUMDIM;  /* STORE NUMBER OF DIM */
00752  2        DO I = 1 TO NUMDIM;  /* STORE DISPLACEMENTS */
00753  2             ARRAYADDR = ARRAYADDR + 2;
00754  3             ARRAYPOS = ARA;
00755  3             CALL POP$STACK;
00756  3             END;
00757  2        CALL PUSH$STACK;  /* NOW PUT ADDRESS OF ARRAY ON STACK */
00758  2        ARA = SAVERA;
00759  2    END CALC$ROW;
00760  1
00761  1
00762  1    CALC$SUB: PROCEDURE;
00763  2        DECLARE
00764  2                ARRAYADDR ADDRESS,
00765  2                ARRAYPOS  BASED ARRAYADDR ADDRESS,
00766  2                I         BYTE,
00767  2                NUMDIM    BYTE,
00768  2                LOCATION  ADDRESS;
00769  2
00770  2        INC$ARRAYADDR: PROCEDURE;
00771  3            ARRAYADDR = ARRAYADDR + 1 + 1;
00772  3        END INC$ARRAYADDR;
00773  2
```

```
00774  2        ARRAYADDR = ARA;
00775  2        CALL POP$STACK;
00776  2        LOCATION = ARA;
00777  2        NUMDIM = ARRAYPOS;
00778  2        DO I = 2 TO NUMDIM;
00779  2             CALL POP$STACK;
00780  3             CALL INC$ARRAYADDR;
00781  3             LOCATION = ARA * ARRAYPOS + LOCATION;
00782  3             END;
00783  2        CALL INC$ARRAYADDR;
00784  2        IF LOCATION >= ARRAYPOS THEN
00785  2             CALL ERROR('S8');
00786  2        ARA = ARRAYADDR + 2 + TIMES4(LOCATION);
00787  2    END CALC$SUB;
00788  1        /*
00789  1             ************************************************************
00790  1             *                                                          *
00791  1             *      STORE PLACES RA IN THE PRT LOCATION REFERENCED      *
00792  1             *      BY RB.  RA MAY CONTAIN A FLOATING POINT NUMBER      *
00793  1             *      OR A REFERENCE TO A STRING.                         *
00794  1             *      IN THE CASE OF A STRING THE FOLLOWING IS ALSO       *
00795  1             *      PERFORMED:                                          *
00796  1             *          (1)  IF THE PRT CELL ALREADY CONTAINS A         *
00797  1             *          REFERENCE TO A STRING IN THE FSA THAT           *
00798  1             *          STRING'S COUNTER IS DECREMENTED AND IF          *
00799  1             *          EQUAL TO 1 THEN THE SPACE IS FREED              *
00800  1             *          (2)  THE NEW STRINGS COUNTER IS INCREMENTED     *
00801  1             *          IF IT IS ALREADY 255 THEN A COPY IS MADE        *
00802  1             *          AND THE NEW COUNTER SET TO 2.                   *
00803  1             *                                                          *
00804  1             ************************************************************
00805  1        */
00806  1
00807  1    STORE: PROCEDURE(TYPE);
00808  2        DECLARE
00809  2                TYPE       BYTE,
00810  2                PTRADDR    ADDRESS,
00811  2                PTR        ADDRESS,
00812  2                STRINGADDR BASED PTRADDR  ADDRESS,
00813  2                COUNTER    BASED PTR      BYTE;
00814  2        CALL SET$DATA$ADDR(RB);
00815  2        IF TYPE THEN  /* STORE STRING */
00816  2             DO;
00817  2                  CALL FLAG$STRING$ADDR(0);  /* SET TEMP STRING OFF */
00818  3             PTRADDR = ARB;  /* CAN WE FREE STRING DESTINATION POINTED TO */
00819  3                  IF IN$FSA(STRINGADDR) THEN   /* IN FSA   */
00820  3                       DO;
00821  3                            PTR = STRINGADDR - 1;
00822  4                            IF(COUNTER := COUNTER - 1) = 1 THEN
00823  4                                 CALL RELEASE(STRINGADDR);
00824  4                       END;
00825  3                  IF IN$FSA(PTR := ARA - 1) THEN   /* INC COUNTER */
00826  3                       DO;
00827  3                       IF COUNTER = 255 THEN  /* ALREADY POINTED TO BY
00828  4                                                 254 VARIABLES */
00829  4                            DO;
00830  4                            PTR = PTR + 1;
00831  5                            CALL MOVE(PTR,ARA := GETSPACE(COUNTER + 1),
00832  5                                      COUNTER + 1);
00833  5                            PTR = ARA - 1;
00834  5                            END;
00835  4                       COUNTER = COUNTER + 1;
00836  4                       END;
00837  3             END;
00838  2        CALL MOVE4(RA,ARB);
00839  2        END STORE;
00840  1        /*
00841  1             ************************************************************
00842  1             *                                                          *
00843  1             *                    BRANCHING ROUTINES                    *
00844  1             *                                                          *
00845  1             ************************************************************
00846  1        */
00847  1
00848  1    UNCOND$BRANCH: PROCEDURE;
00849  2        RC = RC + ARA - 1;
00850  2        CALL POP$STACK;
00851  2    END UNCOND$BRANCH;
00852  1
00853  1
00854  1    COND$BRANCH: PROCEDURE;
00855  2        IF RB$ZERO THEN
00856  2             CALL UNCOND$BRANCH;
00857  2        ELSE
00858  2             CALL POP$STACK;
00859  2        CALL POP$STACK;
00860  2    END COND$BRANCH;
00861  1
00862  1
00863  1    ABSOLUTE$BRANCH: PROCEDURE;
00864  2        CALL STEP$INS$CNT;
00865  2        RC = TWOBYTEOPRAND;
00866  2        RETURN;
00867  2    END ABSOLUTE$BRANCH;
00868  1        /*
00869  1             ************************************************************
00870  1             *                                                          *
```

```
00871  1              *                GLOBAL STRING HANDLING ROUTINES          *
00872  1              *                                                         *
00873  1              ***********************************************************
00874  1          */
00875  1
00876  1      CHECK$STRING$ADDR: PROCEDURE BYTE;
00877  2          RETURN BRA(2);
00878  2      END CHECK$STRING$ADDR;
00879  1
00880  1      STRING$FREE: PROCEDURE;
00881  2          IF CHECK$STRING$ADDR THEN
00882  2              CALL RELEASE(ARA);
00883  2      END STRING$FREE;
00884  1
00885  1
00886  1      GET$STRING$LEN: PROCEDURE(X) BYTE;
00887  2          DECLARE
00888  2                      X       ADDRESS,
00889  2                      A       BASED X     BYTE;
00890  2          IF X = 0 THEN
00891  2              RETURN 0;
00892  2          RETURN A;
00893  2      END GET$STRING$LEN;
00894  1
00895  1      COMP$FIX: PROCEDURE(FLAG);
00896  2          DECLARE FLAG        BYTE,
00897  2                  MINUSONE DATA(81H,80H,0,0);
00898  2          CALL POP$STACK;
00899  2          IF FLAG THEN
00900  2              CALL MOVE4(.MINUSONE,RA);
00901  2          ELSE
00902  2              BRA = 0;
00903  2      END COMP$FIX;
00904  1
00905  1
00906  1      CONCATENATE: PROCEDURE;
00907  2          /*
00908  2              ***********************************************************
00909  2              *                                                         *
00910  2              *    THE STRING POINTED TO BY RA IS CONCATENATED          *
00911  2              *    TO THE STRING POINTED TO BY RB AND THE POINTER       *
00912  2              *    TO THE RESULT IS PLACED IN RB. THE STACK IS POPPED   *
00913  2              *    AND THE RESULT IS FLAGGED AS A TEMPORARY             *
00914  2              *    STRING.                                              *
00915  2              *                                                         *
00916  2              ***********************************************************
00917  2          */
00918  2          DECLARE FIRSTSTRINGLENGTH  BYTE,
00919  2                  SECONDSTRINGLENGTH BYTE,
00920  2                  NEWSTRINGLENGTH    BYTE,
00921  2                  NEWSTRINGADDRESS   ADDRESS,
00922  2                  LENGTH             BASED NEWSTRINGADDRESS BYTE;
00923  2          IF RA$ZERO$ADDRESS THEN  /* IT DOESNT MATTER WHAT RB IS */
00924  2              DO;
00925  2                  CALL POP$STACK;
00926  3                  RETURN;
00927  3              END;
00928  2          IF RB$ZERO$ADDRESS THEN /* AS ABOVE BUT RESULT IS RA */
00929  2              DO;
00930  2                  CALL MOVE$RA$RB;
00931  3                  CALL POP$STACK;
00932  3                  RETURN;
00933  3              END;
00934  2          NEWSTRINGLENGTH = (SECONDSTRINGLENGTH := GETSTRINGLEN(ARA))
00935  2                          + (FIRSTSTRINGLENGTH := GETSTRINGLEN(ARB) + 1);
00936  2          IF CARRY THEN
00937  2              CALL ERROR('SL');
00938  2          CALL MOVE(ARB,NEWSTRINGADDRESS := GETSPACE(NEWSTRINGLENGTH),
00939  2                  FIRSTSTRINGLENGTH);
00940  2          CALL MOVE(ARA + 1,NEWSTRINGADDRESS + FIRSTSTRINGLENGTH,
00941  2                  SECONDSTRINGLENGTH);
00942  2          CALL STRINGFREE;
00943  2          CALL POPSTACK;
00944  2          CALL STRINGFREE;
00945  2          ARA = NEWSTRINGADDRESS;
00946  2          LENGTH = NEWSTRINGLENGTH - 1;
00947  2          CALL FLAG$STRING$ADDR(TRUE);
00948  2      END CONCATENATE;
00949  1
00950  1
00951  1      COMPARE$STRING: PROCEDURE BYTE;
00952  2          /*
00953  2              ***********************************************************
00954  2              *                                                         *
00955  2              *    THE STRING POINTED TO BY RB IS COMPARED TO           *
00956  2              *    THE STRING POINTED TO BY RA.                         *
00957  2              *                      RB RELATION RA                     *
00958  2              *    IF RB < RA THEN RETURN 1                             *
00959  2              *    IF RB > RA THE RETURN 2                              *
00960  2              *    IF RB = RA THEN RETURN 3                             *
00961  2              *    TWO STRINGS ARE EQUAL IF AND ONLY IF THE TWO         *
00962  2              *    STRINGS HAVE THE SAME LENGTH AND CONTAIN             *
00963  2              *    IDENTICAL CHARACTERS. THE ASCII COLLATING            *
00964  2              *    SEQUENCE IS USED TO DETERMINE THE RELATIONSHIP       *
00965  2              *    BETWEEN EQUAL LENGTH STRINGS. IF TWO STRINGS         *
00966  2              *    ARE NOT OF EQUAL LENGTH THE SHORTER IS ALWAYS        *
00967  2              *    LESS THEN THE LONGER ONE. ALL NULL STRINGS ARE       *
00968  2              *    EQUAL AND LESS THEN ANY OTHER STRING.                *
00969  2              *                                                         *
```

```
00970  2          ************************************************************
00971  2       */
00972  2       DECLARE FIRSTSTRING ADDRESS,
00973  2               SECONDSTRING ADDRESS,
00974  2               I           BYTE,
00975  2               TEMPLENGTH  BYTE,
00976  2               CHARSTRING1 BASED FIRSTSTRING BYTE,
00977  2               CHARSTRING2 BASED SECONDSTRING BYTE;
00978  2
00979  2       /* FIRST HANDLE NULL STRINGS REPRESENTED BY RA AND OR RB
00980  2       EQUAL TO ZERO */
00981  2       IF RA$ZERO$ADDRESS THEN
00982  2            SECONDSTRING= RA;
00983  2       ELSE
00984  2            SECONDSTRING = ARA;
00985  2       IF RB$ZERO$ADDRESS THEN
00986  2            FIRSTSTRING = RB;
00987  2       ELSE
00988  2            FIRSTSTRING = ARB;
00989  2       TEMPLENGTH = CHARSTRING1;
00990  2       DO I = 0 TO TEMPLENGTH;
00991  2            IF CHARSTRING1 < CHARSTRING2 THEN
00992  3                 RETURN 1;
00993  3            IF CHARSTRING1 > CHARSTRING2 THEN
00994  3                 RETURN 2;
00995  3            FIRSTSTRING = FIRSTSTRING + 1;
00996  3            SECONDSTRING = SECONDSTRING + 1;
00997  3            END;
00998  2       RETURN 3;
00999  2    END COMPARE$STRING;
01000  1
01001  1    STRING$SEGMENT: PROCEDURE(TYPE);
01002  2       DECLARE
01003  2               LEFT  LIT '0',
01004  2               RIGHT LIT '1',
01005  2               MID   LIT '2';
01006  2
01007  2       DECLARE
01008  2               TYPE    BYTE,
01009  2               TEMPA   ADDRESS,
01010  2               TEMPA2  ADDRESS,
01011  2               LNG     BASED TEMPA BYTE,
01012  2               TEMPB1  BYTE,
01013  2               LNG2    BYTE;
01014  2
01015  2       INC$BRA: PROCEDURE BYTE;
01016  3            RETURN BRA + 1;
01017  3       END INC$BRA;
01018  2
01019  2       TEMPB1 = 0;
01020  2       IF TYPE = MID THEN
01021  2            DO;
01022  2                 CALL FLIP;
01023  3                 IF RA$NEGATIVE OR RA$ZERO THEN
01024  3                      CALL ERROR('SS');
01025  3                 CALL CONV$TO$BIN$ADDR;
01026  3                 TEMPB1 = BRA;
01027  3                 CALL POP$STACK;
01028  3            END;
01029  2        IF RA$NEGATIVE OR (TEMPB1 > GETSTRING$LEN(ARB)) OR RA$ZERO THEN
01030  2            DO;
01031  2                 CALL POP$STACK;
01032  3                 CALL STRINGFREE;
01033  3                 ARA = 0;
01034  3                 RETURN;
01035  3            END;
01036  2       CALL CONV$TO$BIN$ADDR;
01037  2       IF BRA > (LNG2 := GETSTRING$LEN(ARB) - TEMPB1) THEN
01038  2            DO;
01039  2                 IF TYPE=MID THEN
01040  3                      BRA = LNG2 + 1;
01041  3                 ELSE
01042  3                      BRA = LNG2;
01043  3            END;
01044  2       IF TYPE = LEFT THEN
01045  2            TEMPA2 = ARB;
01046  2       ELSE
01047  2            IF TYPE = RIGHT THEN
01048  2                 TEMPA2 = ARB + LNG2 - BRA;
01049  2            ELSE
01050  2                 TEMPA2 = ARB + TEMPB1 - 1;
01051  2       CALL MOVE(TEMPA2,(TEMPA := GETSPACE(INC$BRA)),INC$BRA);
01052  2       LNG = BRA;
01053  2       CALL POP$STACK;
01054  2       CALL STRINGFREE;
01055  2       ARA = TEMPA;
01056  2       CALL FLAG$STRING$ADDR(TRUE);
01057  2    END STRING$SEGMENT;
01058  1
01059  1
01060  1    LOGICAL: PROCEDURE(TYPE);
01061  2       DECLARE
01062  2               TYPE   BYTE,
01063  2               I      BYTE;
01064  2       CALL CONV$TO$BINARY(RA);
01065  2       IF TYPE > 0 THEN
01066  2            CALL CONV$TO$BINARY(RB);
```

```
01068  2         DO I = 0 TO 3;
01069  2           DO CASE TYPE;
01070  3             BRA(I) = NOT BRA(I);
01071  4             BRB(I) = BRA(I) AND BRB(I);
01072  4             BRB(I) = BRA(I) OR BRB(I);
01073  4             BRB(I) = BRA(I) XOR BRB(I);
01074  4           END;
01075  3         END; /* OF DO TWICE */
01076  2         IF TYPE > 0 THEN
01077  2           CALL POP$STACK;
01078  2         CALL CONV$TO$FP(RA);
01079  2       END LOGICAL;
01080  1
01081  1
01082  1       /*
01083  1             ***************************************************
01084  1             *                                                 *
01085  1             *                CONSOLE OUTPUT ROUTINES          *
01086  1             *                                                 *
01087  1             ***************************************************
01088  1       */
01089  1       NUMERIC$OUT: PROCEDURE;
01090  2         /*
01091  2             ***************************************************
01092  2             *                                                 *
01093  2             *    THE FLOATING POINT NUMBER IN RA IS CONVERTED TO   *
01094  2             *    AN ASCII CHARACTER STRING AND THEN PLACED     *
01095  2             *    IN THE WORKBUFFER.  THE LENGTH OF THE STRING  *
01096  2             *    SET TO THE FIRST BYTE OF THE BUFFER           *
01097  2             *                                                 *
01098  2             ***************************************************
01099  2         */
01100  2         DECLARE
01101  2                 I       BYTE;  /* INDEX */
01102  2         CALL FP$OP(FLOD,RA);  /* LOAD FP ACCUM WITH NUMBER FROM RA */
01103  2         CALL FP$OUT(.PRINTWORKAREA(1));  /* CONVERT IT TO ASCII  */
01104  2                 /* RESULT IN PRINTWORKAREA PLUS 1 */
01105  2          I = 0;
01106  2          DO WHILE PRINTWORKAREA(I := I + 1) <> ' ';
01107  2             END;
01108  2          ARA = .PRINTWORKAREA;
01109  2          PRINTWORKAREA = I;
01110  2       END NUMERIC$OUT;
01111  1
01112  1
01113  1       CLEAR$PRINT$BUFF: PROCEDURE;
01114  2         CALL FILL((PRINTBUFFER := PRINTBUFFERLOC),' ',72);
01115  2       END CLEAR$PRINT$BUFF;
01116  1
01117  1
01118  1       DUMP$PRINT$BUFF: PROCEDURE;
01119  2         DECLARE
01120  2                 TEMP  ADDRESS,
01121  2                 CHAR  BASED TEMP BYTE;
01122  2         TEMP=PRINTBUFFEND;
01123  2         DO WHILE CHAR = ' ';
01124  2            TEMP=TEMP - 1;
01125  3         END;
01126  2         CALL CRLF;
01127  2         DO PRINTBUFFER = PRINTBUFFERLOC TO TEMP;
01128  2            CALL PRINTCHAR(PRINTPOS);
01129  3         END;
01130  2         CALL CLEAR$PRINT$BUFF;
01131  2       END DUMP$PRINT$BUFF;
01132  1
01133  1       WRITE$TO$CONSOLE: PROCEDURE;
01134  2         DECLARE
01135  2                 HOLD     ADDRESS,
01136  2                 H        BASED     HOLD    BYTE,
01137  2                 INDEX    BYTE;
01138  2         IF (HOLD := ARA) <> 0 THEN  /* MAY BE NULL STRING */
01139  2           DO INDEX = 1 TO H;
01140  2             PRINTPOS = H(INDEX);
01141  3             IF (PRINTBUFFER := PRINTBUFFER + 1) >
01142  3                                 PRINTBUFFEND THEN
01143  3                   CALL DUMPPRINTBUFF;
01144  3           END;
01145  2       END WRITE$TO$CONSOLE;
01146  1
01147  1
01148  1       /*
01149  1             ***************************************************
01150  1             *                                                 *
01151  1             *      FILE PROCESSING ROUTINES FOR USE WITH CP/M     *
01152  1             *                                                 *
01153  1             ***************************************************
01154  1       */
01155  1       INITIALIZE$DISK$BUFFER: PROCEDURE;
01156  2         CALL FILL(BUFFER,EOFFILLER,128);
01157  2       END INITIALIZE$DISK$BUFFER;
01158  1
01159  1
01160  1       BUFFER$STATUS$BYTE: PROCEDURE BYTE;
01161  2         RETURN FCB(33);
01162  2       END BUFFER$STATUS$BYTE;
01163  1
```

```
01164  1    SET$BUFFER$STATUS$BYTE: PROCEDURE(STATUS);
01165  2        DECLARE STATUS BYTE;
01166  2        FCB(33) = STATUS;
01167  2    END SET$BUFFER$STATUS$BYTE;
01168  1
C1169  1
C1170  1    WRITE$MARK: PROCEDURE BYTE;
01171  2        RETURN BUFFER$STATUS$BYTE;
01172  2    END WRITE$MARK;
01173  1
01174  1
01175  1    SET$WRITE$MARK: PROCEDURE;
01176  2        CALL SET$BUFFER$STATUS$BYTE(BUFFER$STATUS$BYTE OR 01H);
01177  2    END SET$WRITEMARK;
01178  1
C1179  1
C1180  1    CLEAR$WRITE$MARK: PROCEDURE;
01181  2        CALL SET$BUFFER$STATUS$BYTE(BUFFER$STATUS$BYTE AND 0FEH);
01182  2    END CLEAR$WRITE$MARK;
01183  1
01184  1
C1185  1    ACTIVE$BUFFER: PROCEDURE BYTE;
01186  2        RETURN SHR(BUFFER$STATUS$BYTE,1);
01187  2    END ACTIVE$BUFFER;
01188  1
01189  1    SET$BUFFER$INACTIVE: PROCEDURE;
C1190  2        CALL SET$BUFFER$STATUS$BYTE(BUFFER$STATUS$BYTE AND 0FDH);
01191  2    END SET$BUFFER$INACTIVE;
01192  1
01193  1    SET$BUFFER$ACTIVE: PROCEDURE;
01194  2        CALL SET$BUFFER$STATUS$BYTE(BUFFER$STATUS$BYTE OR 02H);
01195  2    END SET$BUFFER$ACTIVE;
C1196  1
01197  1
C1198  1    SET$RANDOM$MODE: PROCEDURE;
C1199  2        CALL SET$BUFFER$STATUS$BYTE(BUFFER$STATUS$BYTE OR 80H);
C1200  2    END SET$RANDOM$MODE;
01201  1
C1202  1    RANDOM$MODE: PROCEDURE BYTE;
01203  2        RETURN RCL(BUFFER$STATUS$BYTE,1);
01204  2    END RANDOM$MODE;
01205  1
01206  1
01207  1    DISK$EOF: PROCEDURE;
01208  2              IF EOFADDR = 0 THEN
01209  2                   CALL ERROR('EF');
C1210  2              RC = EOFADDR + 1;
01211  2              RA = EOFRA;
01212  2              RB = EOFRB;
01213  2              GOTO EOFEXIT;  /* DROP OUT TO OUTER LOOP */;
01214  2    END DISK$EOF;
01215  1
01216  1
01217  1    FILL$FILE$BUFFER: PROCEDURE;
01218  2        DECLARE FLAG BYTE;
C1219  2        IF(FLAG := DISKREAD) = 0 THEN
C1220  2              DO;
01221  2                   CALL SET$BUFFER$ACTIVE;
01222  3                   RETURN;
01223  3              END;
01224  2        IF FLAG = 1 THEN
01225  2              CALL DISK$EOF;
01226  2              RETURN;
01227  2        IF NOT RANDOM$MODE THEN
01228  2              CALL ERROR('CR');
01229  2        CALL INITIALIZE$DISK$BUFFER;
C1230  2        CALL SET$BUFFER$ACTIVE;
01231  2        RETURN;
01232  2    END FILL$FILE$BUFFER;
01233  1
01234  1
01235  1    WRITE$DISK$IF$REQ: PROCEDURE;
01236  2        IF WRITE$MARK THEN
01237  2        DO;
01238  2              IF DISKWRITE <> 0 THEN
01239  3                   CALL ERROR('DW');
C1240  3              CALL CLEAR$WRITE$MARK;
01241  3              IF RANDOM$MODE THEN
01242  3                   CALL SET$BUFFER$INACTIVE;
01243  3              ELSE
01244  3                   CALL INITIALIZE$DISK$BUFFER;
01245  3        END;
01246  2        RECORD$POINTER = BUFFER;
01247  2    END WRITE$DISK$IF$REQ;
C1248  1
C1249  1
01250  1    AT$END$DISK$BUFFER: PROCEDURE BYTE;
01251  2        RETURN (RECORD$POINTER := RECORD$POINTER + 1) >= BUFFER$END;
01252  2    END AT$END$DISK$BUFFER;
01253  1
01254  1    VAR$BLOCK$SIZE: PROCEDURE BYTE;
01255  2        RETURN BLOCKSIZE <> 0;
01256  2    END VAR$BLOCKSIZE;
01257  1
01258  1
C1259  1
```

```
01260  1    STORE$REC$PTR: PROCEDURE;
01261  2        FCBADD(18)=RECORD$POINTER;
01262  2    END STORE$REC$PTR;
01263  1
01264  1    WRITE$A$BYTE: PROCEDURE(CHAR);
01265  2        DECLARE CHAR BYTE;
01266  2        IF VAR$BLOCK$SIZE AND (BYTESWRITTEN := BYTESWRITTEN + 1)
01267  2                > BLOCKSIZE THEN
01268  2            CALL ERROR('ER');
01269  2        IF AT$END$DISK$BUFFER THEN
01270  2            CALL WRITE$DISK$IF$REQ;
01271  2        IF NOT ACTIVE$BUFFER AND RANDOM$MODE THEN
01272  2            DO;
01273  2                CALL FILL$FILE$BUFFER;
01274  3                FCB(32) = FCB(32) - 1;  /* RESET RECORD NO */
01275  3            END;
01276  2        NEXTDISKCHAR = CHAR;
01277  2        CALL SET$WRITE$MARK;
01278  2    END WRITE$A$BYTE;
01279  1
01280  1
01281  1    GET$FILE$NUMBER: PROCEDURE BYTE;
01282  2        IF BRA > NUMFILES THEN
01283  2            CALL ERROR('MF');
01284  2        RETURN BRA;
01285  2    END GET$FILE$NUMBER;
01286  1
01287  1
01288  1    SET$FILE$ADDR: PROCEDURE;
01289  2        DECLARE CURRENTFILE BYTE;
01290  2        FILEADDR = FILES(CURRENTFILE := GET$FILE$NUMBER);
01291  2        EOFADDR = EOFBRANCH(CURRENTFILE);
01292  2    END SET$FILE$ADDR;
01293  1
01294  1
01295  1    SET$FILE$POINTERS: PROCEDURE;
01296  2        BUFFER$END = (BUFFER := FILEADDR + 38) + 128;
01297  2        RECORDPOINTER = FCBADD(18);
01298  2        BLOCKSIZE = FCBADD(17);
01299  2        CALL SETDMA;
01300  2    END SET$FILE$POINTERS;
01301  1
01302  1
01303  1    SETUP$FILE$EXTENT: PROCEDURE;
01304  2        IF OPEN = 255 THEN
01305  2            DO;
01306  2                IF MAKE = 255 THEN
01307  3                    CALL ERROR('ME');
01308  3            END;
01309  2    END SETUP$FILE$EXTENT;
01310  1
01311  1
01312  1    DISK$OPEN: PROCEDURE;
01313  2        /*OPENS THE FILE - RA CONTAINS THE ADDRESS OF THE FILE NAME
01314  2        AND RB CONTAINS THE BLOCK SIZE.
01315  2        THE ARRAY FILES WILL HOLD THE ADDRESS OF THE FILE CONTROL BLOCK
01316  2        IN THE FSA.  THE FCB IS FOLLOWED BY 3 FLAGS - BLOCKSIZE(ADDR)
01317  2        RECORD POINTER(ADDR), WRITE FLAG(BYTE).  THIS IS FOLLOWED BY THE
01318  2        128 BYTE BUFFER TO DO FILE I/O.*/
01319  2
01320  2        DECLARE
01321  2                FILENAME ADDRESS,
01322  2                NEXTFILE BYTE,
01323  2                BUFF     ADDRESS,
01324  2                CHAR     BASED BUFF BYTE,
01325  2                I        BYTE,
01326  2                J        BYTE;
01327  2
01328  2        INC$J: PROCEDURE BYTE;
01329  3            RETURN (J := J + 1);
01330  3        END INC$J;
01331  2
01332  2        NEXTFILE = 0;
01333  2        DO WHILE FILES(NEXTFILE := NEXTFILE + 1) <> 0;
01334  2            END;
01335  2        FILEADDR,FILES(NEXTFILE) = GETSPACE(166);
01336  2        BUFFER = FILEADDR + 38;
01337  2        CALL SETDMA;
01338  2        CALL FILL((FILENAME:=FILEADDR+1),' ',11);
01339  2        BUFF=ARA;
01340  2        IF CHAR(2) = ':' THEN
01341  2            DO;
01342  2                FCB = CHAR(1) AND 0FH;
01343  3                I = CHAR - 2;
01344  3                BUFF = BUFF + 2;
01345  3            END;
01346  2        ELSE
01347  2            I = CHAR;
01348  2        IF I > 12 THEN
01349  2            I = 12;
01350  2        BUFF=BUFF+1;
01351  2        J = 255;
01352  2        DO WHILE(CHAR(INC$J) <> '.') AND (J < I);
01353  2            END;
01354  2        CALL MOVE(BUFF,FILENAME,J);
01355  2        IF I > INC$J THEN
01356  2            CALL MOVE (.CHAR(J),FILENAME + 8, I - J);
01357  2        CALL SETUP$FILE$EXTENT;
```

```
01358  2        FCBADD(18)=FILEADDR+256;
01359  2        CALL POP$STACK;
01360  2        FCBADD(17) = ARA;
01361  2        CALL POP$STACK;
01362  2    END DISK$OPEN;
01363  1
01364  1
01365  1    SET$EOF$STACK: PROCEDURE;
01366  2         EOFRA = RA;
01367  2         EOFRB = RB;
01368  2    END SET$EOF$STACK;
01369  1
01370  1    SETUP$DISK$IC: PROCEDURE;
01371  2
01372  2        CALL SET$FILE$ADDR;
01373  2        IF FILEADDR = 0 THEN
01374  2             CALL ERROR('FU');
01375  2        CALL SET$FILE$POINTERS;
01376  2        BYTES$WRITTEN=0;
01377  2        FIRSTFIELD = TRUE;
01378  2        CALL POP$STACK;
01379  2    END SETUP$DISK$IC;
01380  1
01381  1
01382  1    RANDOM$SETUP: PROCEDURE;
01383  2        DECLARE
01384  2                  BYTECOUNT ADDRESS,
01385  2                  RECORD    ADDRESS,
01386  2                  EXTENT    BYTE;
01387  2
01388  2        IF NOT VAR$BLOCK$SIZE THEN
01389  2             CALL ERROR('RU');
01390  2        IF RA$ZERO OR RA$NEGATIVE THEN
01391  2             CALL ERROR('IR');
01392  2        CALL CONV$TO$BIN$ADDR;
01393  2        ARA = ARA - 1;
01394  2        CALL SET$RANDOM$MODE;
01395  2        CALL SET$BUFFER$INACTIVE;
01396  2        CALL WRITE$DISK$IF$REQ;
01397  2         BYTECOUNT = BLOCKSIZE * ARA;
01398  2        RECORDPOINTER = (BYTECOUNT AND 7FH) + BUFFER - 1;
01399  2        CALL STORE$REC$PTR;
01400  2        RECORD = SHR(BYTECOUNT,7);
01401  2        EXTENT = SHR(RECORD,7);
01402  2        IF EXTENT<>FCB(12) THEN
01403  2             DO;
01404  2                  IF CLOSE = 255 THEN
01405  3                       CALL ERROR('CE');
01406  3                  FCB(12) = EXTENT;
01407  3                  CALL SETUP$FILE$EXTENT;
01408  3             END;
01409  2        FCB(32) = LOW(RECORD) AND 7FH;
01410  2        CALL POP$STACK;
01411  2    END RANDOM$SETUP;
01412  1
01413  1
01414  1    GET$DISK$CHAR: PROCEDURE BYTE;
01415  2         IF AT$END$DISK$BUFFER THEN
01416  2             DO;
01417  2                  CALL WRITE$DISK$IF$REQ;
01418  3                  CALL FILL$FILE$BUFFER;
01419  3             END;
01420  2        IF NOT ACTIVE$BUFFER THEN
01421  2             CALL FILL$FILE$BUFFER;
01422  2        IF NEXTDISKCHAR = EOFFILLER THEN
01423  2             CALL DISK$EOF;
01424  2        RETURN NEXTDISKCHAR;
01425  2    END GET$DISK$CHAR;
01426  1
01427  1
01428  1    WRITE$TO$FILE: PROCEDURE(TYPE);
01429  2        /* TYPE 0 MEANS WRITE A NUMBER, 1 MEANS A STRING*/
01430  2        DECLARE
01431  2                  I       BYTE,
01432  2                  POINT   ADDRESS,
01433  2                  CHAR    BASED     POINT  BYTE,
01434  2                  CCUNT   BYTE,
01435  2                  TYPE    BYTE,
01436  2                  NUMERIC LIT       '0',
01437  2                  STRING  LIT       '1';
01438  2
01439  2        INC$POINT: PROCEDURE;
01440  3             POINT = POINT + 1;
01441  3        END INC$POINT;
01442  2
01443  2        IF TYPE = NUMERIC THEN
01444  2             CALL NUMERICOUT;
01445  2        IF NOT FIRSTFIELD THEN
01446  2             CALL WRITE$A$BYTE(',');
01447  2        ELSE
01448  2             FIRSTFIELD = FALSE;
01449  2        POINT = ARA;
01450  2        CCUNT = CHAR;
01451  2        IF TYPE = NUMERIC THEN
01452  2             COUNT = COUNT + 1;
01453  2        ELSE
01454  2             CALL WRITE$A$BYTE(QUOTE);
```

```
01455  2        CALL INC$POINT;
01456  2        DO I = 1 TO COUNT;
01457  2             IF CHAR = QUOTE THEN
01458  3                  CALL ERROR('QE');
01459  3             CALL WRITE$A$BYTE(CHAR);
01460  3             CALL INC$POINT;
01461  3             END;
01462  2        IF TYPE = STRING THEN
01463  2             DO;
01464  2                  CALL WRITE$A$BYTE(QUOTE);
01465  3                  CALL STRING$FREE;
01466  3             END;
01467  2        CALL POP$STACK;
01468  2   END WRITE$TO$FILE;
01469  1
01470  1
01471  1   DISK$CLOSE: PROCEDURE;
01472  2        CALL SET$FILE$POINTERS;
01473  2        CALL WRITE$DISK$IF$REQ;
01474  2        IF CLOSE = 255 THEN
01475  2             CALL ERROR('CE');
01476  2        CALL RELEASE(FILEADDR);
01477  2   END DISK$CLOSE;
01478  1
01479  1   CLOSEFILES:  PROCEDURE;
01480  2        DECLARE I BYTE;
01481  2        I = 0;
01482  2        DO WHILE(I:=I+1) < NUMFILES;
01483  2             IF(FILEADDR := FILES(I)) <> 0 THEN
01484  3                  CALL DISKCLOSE;
01485  3             END;
01486  2   END CLOSEFILES;
01487  1
01488  1        /*
01489  1             ************************************************************
01490  1             *                                                          *
01491  1             *                  ROUTINE TO EXIT INTERP                  *
01492  1             *                                                          *
01493  1             ************************************************************
01494  1        */
01495  1   EXIT$INTERP: PROCEDURE;
01496  2        CALL CLOSEFILES;
01497  2        CALL DUMP$PRINT$BUFF;
01498  2        CALL CRLF;
01499  2        CALL MON3;
01500  2   END EXIT$INTERP;
01501  1
01502  1
01503  1        /*
01504  1             ************************************************************
01505  1             *                                                          *
01506  1             *              GENERALIZED INPUT ROUTINES                  *
01507  1             *                                                          *
01508  1             ************************************************************
01509  1        */
01510  1
01511  1   CONSOLE$READ: PROCEDURE;
01512  2        CALL PRINTCHAR(WHAT);
01513  2        CALL PRINTCHAR(' ');
01514  2        CALL READ(.INPUTBUFFER);
01515  2        IF SPACE(1) = CONTZ THEN
01516  2             CALL EXIT$INTERP;
01517  2        CONBUFFPTR = .SPACE;
01518  2        SPACE(SPACE+1)=EOLCHAR;
01519  2   END CONSOLE$READ;
01520  1
01521  1   MORE$CON$INPUT: PROCEDURE BYTE;
01522  2        RETURN CONBUFFPTR < .SPACE(SPACE);
01523  2   END MORE$CON$INPUT;
01524  1
01525  1
01526  1   CONSOLE$INPUT$ERROR: PROCEDURE;
01527  2        RC = REREADADDR;  /* RESET PROGRAM COUNTER */
01528  2        CALL WARNING('II');
01529  2        GOTO ERROR$EXIT;  /* RETURN TO OUTER LEVEL */
01530  2   END CONSOLE$INPUT$ERROR;
01531  1
01532  1
01533  1   GET$DATA$CHAR: PROCEDURE BYTE;
01534  2        DECLARE CHAR BASED DATAAREAPTR BYTE;
01535  2        IF(DATAAREAPTR := DATAAREAPTR + 1) >= SB THEN
01536  2             CALL ERROR('OD');
01537  2        RETURN CHAR;
01538  2   END GET$DATA$CHAR;
01539  1
01540  1
01541  1   GET$CON$CHAR: PROCEDURE BYTE;
01542  2        DECLARE CHAR BASED CONBUFFPTR BYTE;
01543  2        CONBUFFPTR = CONBUFFPTR + 1;
01544  2        RETURN CHAR;
01545  2   END GET$CON$CHAR;
01546  1
01547  1
01548  1   NEXT$INPUT$CHAR: PROCEDURE BYTE;
01549  2        IF INPUTTYPE = 0 THEN
01550  2             DO FOREVER;
01551  2                  IF(SPACE(INPUTINDEX):= GETDISKCHAR) = LF THEN
01552  3                       DO;
```

```
01553  3                            IF VAR$BLOCKSIZE THEN
01554  4                                CALL ERROR('RE');
01555  4                        END;
01556  3                    ELSE
01557  3                        RETURN NEXTDISKCHAR;
01558  3                END;
01559  2            IF INPUTTYPE = 1 THEN
01560  2                RETURN GETCONCHAR;
01561  2            IF INPUTTYPE = 2 THEN
01562  2                RETURN GETDATACHAR;
01563  2        END NEXT$INPUT$CHAR;
01564  1
01565  1
01566  1        COUNT$INPUT: PROCEDURE;
01567  2            DECLARE
01568  2                    HOLD  BYTE,
01569  2                    DELIM BYTE;
01570  2            INPUT$INDEX = 0;
01571  2            DO WHILE (HOLD := NEXT$INPUT$CHAR) = ' ';
01572  2                END;
01573  2            IF INPUTTYPE = 0 THEN
01574  2                INPUTPTR = .SPACE;
01575  2            IF INPUTTYPE = 1 THEN
01576  2                INPUTPTR = CONBUFFPTR;
01577  2
01578  2            IF INPUTTYPE =2 THEN
01579  2                INPUTPTR = DATAAREAPTR;
01580  2            IF HOLD <> QUOTE THEN
01581  2                DELIM = ',';
01582  2            ELSE
01583  2                DO;
01584  2                    DELIM = QUOTE;
01585  3                    IF INPUTTYPE <> 0 THEN
01586  3                        INPUTPTR = INPUTPTR + 1;
01587  3                    HOLD = NEXT$INPUT$CHAR;
01588  3                END;
01589  2            DO WHILE (HOLD <> DELIM) AND (HOLD <> EOLCHAR);
01590  2                INPUTINDEX = INPUTINDEX + 1;
01591  3                HOLD = NEXT$INPUT$CHAR;
01592  3                END;
01593  2            IF DELIM = QUOTE THEN
01594  2                DO WHILE((HOLD := NEXT$INPUT$CHAR) <> ',') AND (HOLD <> EOLCHAR);
01595  2                    END;
01596  2            CALL PUSH$STACK;
01597  2        END COUNT$INPUT;
01598  1
01599  1
01600  1        GET$STRING$FIELD: PROCEDURE;
01601  2            DECLARE
01602  2                    TEMP ADDRESS,
01603  2                    LNG  BASED TEMP BYTE;
01604  2            CALL COUNT$INPUT;
01605  2            CALL MOVE(INPUTPTR,(TEMP:=GETSPACE(INPUTINDEX + 1))+1,INPUTINDEX);
01606  2            ARA = TEMP;
01607  2            CALL FLAG$STRING$ADDR(0);
01608  2            LNG = INPUTINDEX;  /* SET LENGTH IN NEW STRING */
01609  2        END GET$STRING$FIELD;
01610  1
01611  1
01612  1        GET$NUMERIC$FIELD: PROCEDURE;
01613  2            CALL COUNT$INPUT;
01614  2            CALL FP$INPUT(INPUTINDEX,INPUTPTR);
01615  2            CALL FP$CP$RETURN(9,RA);
01616  2            CALL CHECK$OVERFLOW;
01617  2        END GET$NUMERIC$FIELD;
01618  1
01619  1
01620  1
01621  1            /*
01622  1                 ************************************************************
01623  1                 *                                                          *
01624  1                 *          INTERPRETER INITIALIZATION ROUTINES             *
01625  1                 *                                                          *
01626  1                 ************************************************************
01627  1            */
01628  1
01629  1
01630  1        INITIALIZE$EXECUTE: PROCEDURE;
01631  2            GET$PARAMETERS: PROCEDURE;
01632  3                DECLARE POINTER ADDRESS INITIAL(08F6H), /*2 LESS THAN PARM LOC*/
01633  3                        PARM    BASED POINTER ADDRESS;
01634  3
01635  3                    NEXT: PROCEDURE ADDRESS;
01636  4                        POINTER=POINTER+2;
01637  4                        RETURN PARM;
01638  4                    END NEXT;
01639  3
01640  3                MCD,RC = NEXT;
01641  3                DATAAREAPTR = (MDA := NEXT) - 1;
01642  3                MPR=NEXT;
01643  3                MBASE,ST = (SB := NEXT) + NRSTACK;
01644  3                RA = (RB := SB) + 4;
01645  3            END GET$PARAMETERS;
01646  2
01647  2            INITMEM: PROCEDURE;
01648  3                DECLARE BASE ADDRESS,
01649  3                        A BASED BASE ADDRESS,
01650  3                        TOP BASED SYSBEGIN ADDRESS;
```

```
01651  3          CALL MOVE(BUILDTOP,.MEMORY,MPR-.MEMORY);
01652  3          CALL FILL(MPR,0,MBASE-MPR);
01653  3          BASE=ST;
01654  3          A=TOP-4;
01655  3          A(1),A(2) = 0;
01656  3          BASE=A;
01657  3          A = 0;
01658  3          A(1) = ST;
01659  3      END INITMEM;
01660  2
01661  2
01662  2      CALL GET$PARAMETERS;
01663  2      CALL INITMEM;
01664  2      CALL FILL(.FILES,0,TIMES4(NUMFILES));
01665  2      CALL CLEAR$PRINT$BUFF;
01666  2  END INITIALIZE$EXECUTE;
01667  1
01668  1
01669  1   /* ***** EXECUTIVE ROUTINE STARTS HERE ***** */
01670  1      /*
01671  1           ************************************************************
01672  1           *                                                          *
01673  1           ************************************************************
01674  1      */
01675  1  EXECUTE:  PROCEDURE;
01676  2       DO FOREVER;
01677  2      IF ROL(C,1) THEN     /* MUST BE LIT OR LIT-LOD*/
01678  3      DO;
01679  3           CALL PUSH$STACK;
01680  4           BRA=C(1);          /* LOAD IN REVERSE ORDER */
01681  4           BRA(1)= C AND 3FH;
01682  4           IF ROL(C,2) THEN CALL LOAD$RA;   /*LIT-LOD*/
01683  4           CALL STEP$INS$CNT;
01684  4           END;
01685  3      ELSE
01686  3           DO CASE C;
01687  3
01688  3   /*0  FAD: RB = RA+ RB  */
01689  3        CALL TWO$VALUE$OPS(FADD);
01690  4
01691  4   /*1  FMI  RB = RB-RA; */
01692  4        DO;
01693  4             CALL FLIP;
01694  5        CALL TWO$VALUE$OPS(FSUB);
01695  5        END;
01696  4
01697  4   /*2 FMU  RB= RA*RB     */
01698  4        CALL TWO$VALUE$OPS(FMUL);
01699  4
01700  4   /*3  FDI  RB = RA/RB  */
01701  4        DO;
01702  4             IF RA$ZERO THEN
01703  5                  CALL WARNING('DZ');
01704  5             CALL FLIP;
01705  5             CALL TWO$VALUE$OPS(FDIV);
01706  5        END;
01707  4
01708  4   /*4  EXP  RA=RB**RA     */
01709  4        DO;
01710  4             IF RB$ZERO THEN
01711  5                  CALL COMP$FIX(RA$ZERO);
01712  5             ELSE
01713  5                  IF RB$NEGATIVE THEN
01714  5                       CALL ERROR('NE');
01715  5                  ELSE
01716  5                       DO;
01717  5                            CALL FP$OP(FLOD,RB);
01718  6                            CALL FP$OP(LOG,0);
01719  6                            CALL FP$OP(FMUL,RA);
01720  6                            CALL FP$OP$RETURN(EXP,RB);
01721  6                            CALL POP$STACK;
01722  6                            CALL CHECK$OVERFLOW;
01723  6                       END;
01724  5        END;
01725  4
01726  4   /* 5 LSS, LESS THEN */
01727  4        CALL COMP$FIX(COMPARE$FP=1);
01728  4
01729  4   /* 6 GTR, GREATER THEN */
01730  4        CALL COMP$FIX(COMPARE$FP=2);
01731  4
01732  4   /* 7  EQU, EQUAL TO */
01733  4        CALL COMP$FIX(COMPARE$FP=3);
01734  4
01735  4   /* 8  NEQ, NOT EQUAL TO */
01736  4        CALL COMP$FIX(NOT(COMPARE$FP=3));
01737  4
01738  4   /* 9  GEQ, GREATER THEN OR EQUAL TO */
01739  4        CALL COMP$FIX(NOT(COMPARE$FP=1));
01740  4
01741  4   /*10  LEQ, LESS THEN OR EQUAL TO */
01742  4        CALL COMP$FIX(NOT(COMPARE$FP=2));
01743  4
01744  4   /*11  NOT*/
01745  4        CALL LOGICAL(0);
01746  4
01747  4   /*12  AND*/
```

```
01748  4              CALL LOGICAL(1);
01749  4
01750  4     /*13  BOR */
01751  4              CALL LOGICAL(2);
01752  4
01753  4     /* 14 LOD*/
01754  4                   CALL LOAD$RA;
01755  4
01756  4     /* 15 STO */
01757  4              DO;
01758  4                   CALL STORE(0);
01759  5                   CALL MOVE$RA$RB;
01760  5                   CALL POP$STACK;
01761  5              END;
01762  4
01763  4     /* 16 XIT */
01764  4              RETURN;
01765  4
01766  4     /* 17 DEL */
01767  4              CALL POP$STACK;
01768  4
01769  4     /* 18 DUP */
01770  4              DO;
01771  4                   CALL PUSH$STACK;
01772  5              CALL MOVE$RB$RA;
01773  5            END;
01774  4
01775  4     /* 19 XCH */
01776  4              CALL FLIP;
01777  4
01778  4     /* 20 STD */
01779  4              DO;
01780  4                   CALL STORE(0);
01781  5                   CALL POP$STACK;
01782  5                   CALL POP$STACK;
01783  5              END;
01784  4
01785  4     /* 21 SLT */
01786  4              CALL COMP$FIX(COMPARE$STRING = 1);
01787  4
01788  4     /* 22 SGT */
01789  4              CALL COMP$FIX(COMPARE$STRING = 2);
01790  4
01791  4     /* 23 SEQ */
01792  4              CALL COMP$FIX(COMPARE$STRING = 3);
01793  4
01794  4     /* 24 SNE */
01795  4              CALL COMP$FIX(NOT(COMPARE$STRING = 3));
01796  4
01797  4     /* 25 SGE */
01798  4              CALL COMP$FIX(NOT(COMPARE$STRING = 1));
01799  4     /* 26 SLE */
01800  4              CALL COMP$FIX(NOT(COMPARE$STRING = 2));
01801  4
01802  4     /* 27  STS */
01803  4              DO;
01804  4                   CALL STORE(1);
01805  5                   CALL POP$STACK;
01806  5                   CALL POP$STACK;
01807  5            END;
01808  4
01809  4     /* 28 ILS */
01810  4              DO;
01811  4                   CALL PUSH$STACK;
01812  5                   CALL STEP$INS$CNT;
01813  5                   RC = (ARA := RC) + C;
01814  5                   CALL FLAG$STRING$ADDR(FALSE);
01815  5              END;
01816  4
01817  4     /* 29 CAT */
01818  4              CALL CONCATENATE;
01819  4     /* 30 PRO */
01820  4              DO;
01821  4                   CALL STEP$INS$CNT;
01822  5                   CALL PUSH$STACK;
01823  5                   ARA = RC + 2;
01824  5                   RC = TWCBYTEOPRAND;
01825  5              END;
01826  4
01827  4     /* 31 RTN */
01828  4              DO;
01829  4                   RC = ARA - 1;
01830  5                   CALL POP$STACK;
01831  5              END;
01832  4
01833  4     /*32 ROW, CALCULATES SPACE REQUIREMENTS FOR ARRAYS*/
01834  4              CALL CALC$ROW;
01835  4
01836  4     /* 33, SUB */
01837  4     /*    SUB,CALCULATES SUBSCRIPT ADDRESSES */
01838  4              CALL CALC$SUB;
01839  4
01840  4
01841  4     /* RDV  READS A NUMBER FROM THE CONSOLE  */
01842  4              DO;
01843  4                   IF NOT MORE$CON$INPUT THEN
01844  5                        CALL CONSOLE$INPUT$ERROR;
```

```
01845  5                    CALL GET$NUMERIC$FIELD;
01846  5               END;
01847  4
01848  4     /* 35, WRV : PRINTS THE NUMBER ON THE TOP OF THE STACK */
01849  4               DO;
01850  4                    CALL NUMERIC$OUT;
01851  5                    CALL WRITE$TO$CONSOLE;
01852  5                    CALL POP$STACK;
01853  5               END;
01854  4
01855  4     /* 36 WST: PRINTS THE STRING WHOSE ADDRESS IS ON TOPOF THE STACK*/
01856  4               DO;
01857  4                    CALL WRITE$TO$CONSOLE;
01858  5                    CALL STRING$FREE;
01859  5                    CALL POP$STACK;
01860  5               END;
01861  4
01862  4     /* 37, RRF */
01863  4        /* RRF - PROCEDURE TO READY A RANDOM BLOCK */
01864  4               DO;
01865  4                    CALL SETUP$DISK$IO;
01866  5                    CALL RANDOM$SETUP;
01867  5                    CALL SET$EOF$STACK;
01868  5               END;
01869  4
01870  4     /* 38, RDB */
01871  4        /* RDB - READY NEXT SEQUENTIAL BLOCK */
01872  4               DO;
01873  4                    CALL SETUP$DISK$IO;
01874  5                    CALL SET$EOF$STACK;
01875  5               END;
01876  4
01877  4     /* 39, ECR */
01878  4               IF MORE$CON$INPUT THEN
01879  4                    CALL CONSOLE$INPUT$ERROR;
01880  4
01881  4     /* 4C, OUT */
01882  4               DO;
01883  4                    OUTPUT: PROCEDURE(PORT,VALUE);
01884  6                         DECLARE
01885  6                                   PORT  BYTE,
01886  6                                   VALUE BYTE;
01887  6                         GOTO PORTOUT;
01888  6                    END OUTPUT;
01889  5
01890  5                    CALL OUTPUT(BRA,BRB);
01891  5                    CALL POP$STACK;
01892  5                    CALL POP$STACK;
01893  5               END;
01894  4
01895  4        /*41 RDN - READ A NUMBER FROM DISK*/
01896  4               DO;
01897  4                    INPUTTYPE = 0;
01898  5                    CALL GET$NUMERIC$FIELD;
01899  5               END;
01900  4
01901  4        /*42 RDS - READ A STRING FROM DISK*/
01902  4               DO;
01903  4                    INPUTTYPE = 0;
01904  5                    CALL GET$STRING$FIELD;
01905  5               END;
01906  4
01907  4        /*43 WRN WRITE A NUMBER TO DISK*/
01908  4        CALL WRITE$TO$FILE(0);
01909  4
01910  4        /*44 WRS - WRITE A STRING TO DISK */
01911  4        CALL WRITE$TO$FILE(1);
01912  4
01913  4     /* 45, OPN */
01914  4     /*OPN:   PROCEDURE TO CREATE FCBS FOR ALL INPUT FILES */
01915  4             CALL DISK$OPEN;
01916  4
01917  4     /* 46 CON */
01918  4               DO;
01919  4                    CALL PUSH$STACK;
01920  5                    CALL STEP$INS$CNT;
01921  5                    CALL MOVE4(TWOBYTEOPRAND,RA);
01922  5                    CALL STEP$INS$CNT;
01923  5               END;
01924  4
01925  4     /* 47, RST: PUTS POINTER TO THE BEGINNING OF THE DATA AREA*/
01926  4                    DATAAREAPTR = MOA - 1;
01927  4
01928  4     /*48  NEG, NEGATIVE */
01929  4               CALL ONE$VALUE$OPS(FCHS);
01930  4
01931  4     /* 49 , RES : READ STRING */
01932  4               DO;
01933  4                    IF NOT MORE$CON$INPUT THEN
01934  5                         CALL CONSOLE$INPUT$ERROR;
01935  5                    CALL GET$STRING$FIELD;
01936  5               END;
01937  4
01938  4     /* 50 NOP */
01939  4     ;
01940  4
01941  4     /* 51 DAT */
01942  4      :
```

```
01943  4
01944  4     /* 52 DBF */
01945  4           CALL DUMPPRINTBUFF;
01946  4
01947  4     /* 53 NSP */
01948  4           DO;
01949  4                DECLARE I BYTE,
01950  5                      POSITION DATA(TABPOS1,TABPOS2,TABPOS3,TABPOS4,
01951  5                      PRINTBUFFEND);
01952  5                I=0;
01953  5                DO WHILE PRINTBUFFER > POSITION(I);
01954  5                     I = I + 1;
01955  6                     END;
01956  5                IF I = 4 THEN
01957  5                     CALL DUMP$PRINT$BUFF;
01958  5                ELSE
01959  5                     PRINTBUFFER = POSITION(I);
01960  5           END;
01961  4
01962  4     /* 54 BRS */
01963  4           CALL ABSOLUTE$BRANCH;
01964  4
01965  4     /* 55 BRC */
01966  4           DO;
01967  4                IF RA$ZERO THEN
01968  5                     CALL ABSOLUTE$BRANCH;
01969  5                ELSE
01970  5                     RC = RC + 1 + 1;
01971  5                CALL POP$STACK;
01972  5           END;
01973  4
01974  4     /* 56 BFC */
01975  4           CALL COND$BRANCH;
01976  4
01977  4     /* 57 BFN */
01978  4           CALL UNCOND$BRANCH;
01979  4
01980  4     /* 58 CBA */
01981  4           CALL CONV$TO$BINARY(RA);
01982  4
01983  4     /* 59 RCN */
01984  4           DO;
01985  4                INPUTTYPE = 1;
01986  5                REREADADDR = RC;
01987  5                CALL CONSOLE$READ;
01988  5           END;
01989  4
01990  4     /* 60 DRS READ STRING FROM DATA AREA */
01991  4           DO;
01992  4                INPUTTYPE = 2;
01993  5                CALL GET$STRING$FIELD;
01994  5           END;
01995  4
01996  4     /* 61 DRF READ F/P NUMBER FROM DATA AREA */
01997  4           DO;
01998  4                INPUTTYPE = 2;
01999  5                CALL GET$NUMERIC$FIELD;
02000  5           END;
02001  4
02002  4        /*62 EDR - END OF RECORD FOR READ*/
02003  4        /*ADVANCES TO NEXT LINE FEED*/
02004  4           DO;
02005  4                IF VAR$BLOCK$SIZE THEN
02006  5                     DO WHILE GET$DISK$CHAR <> LF;
02007  5                          END;
02008  5                CALL STORE$REC$PTR;
02009  5           END;
02010  4
02011  4        /*63 EOW - END OF RECORD FOR WRITE*/
02012  4           DO;
02013  4                IF VAR$BLOCK$SIZE THEN
02014  5                     DO WHILE BYTES$WRITTEN < (BLOCKSIZE - 2);
02015  5                          CALL WRITE$A$BYTE(' ');
02016  6                          END;
02017  5                CALL WRITE$A$BYTE(CR);
02018  5                CALL WRITE$A$BYTE(LF);
02019  5                CALL STORE$REC$PTR;
02020  5           END;
02021  4        /*64 CLS - CLOSE A FILE*/
02022  4           DO;
02023  4                CALL SET$FILE$ADDR;
02024  5                CALL DISK$CLOSE;
02025  5                FILES(BRA) = 0;
02026  5                ECFBRANCH(BRA) = 0;
02027  5                CALL POP$STACK;
02028  5           END;
02029  4
02030  4     /* 65 ABSOLUTE */
02031  4           BRA(1) = BRA(1) AND 7FH;
02032  4
02033  4     /* 66 INTEGER */
02034  4           DO;
02035  4                CALL CONV$TO$BINARY(RA);
02036  5                CALL CONV$TO$FP(RA);
02037  5           END;
02038  4
```

```
02039  4     /* 67 RANDOM NUMBER GENERATOR */
C2040  4           DO;
02041  4                DECLARE SEED BASED SEEDLOC ADDRESS,
02042  5                          SCALE DATA(90H,7FH,0FFH,0);
02043  5                RANDOM: PROCEDURE;
02044  6                     GOTO RANDOMLOC;
02045  6                END RANDOM;
02046  5
02047  5                CALL RANDOM;
02048  5                CALL PUSH$STACK;
C2049  5                CALL MOVE4(.SCALE,RA);
C2050  5                CALL PUSH$STACK;
02051  5                CALL FLOAT$ADDR(SEED);
02052  5                CALL TWO$VALUE$OPS(FDIV);
02053  5           END;
02054  4
C2055  4     /* 68 SGN */
02056  4           DO;
02057  4                DECLARE FLAG BYTE;
02058  5                FLAG = RA$NEGATIVE;
02059  5                CALL COMP$FIX(NOT RA$ZERO);
C2060  5           IF FLAG THEN
02061  5                     CALL ONE$VALUE$OPS(FCHS);
02062  5           END;
02063  4
02064  4     /* 69 SINE */
02065  4           CALL ONE$VALUE$OPS(SIN);
02066  4
02067  4     /* 70 COSINE */
02068  4           CALL ONE$VALUE$OPS(COS);
C2069  4
C2070  4     /* 71 ARCTANGENT */
02071  4           CALL ONE$VALUE$OPS(ATAN);
02072  4
C2073  4     /* 72 TANGENT */
02074  4           DO;
02075  4                CALL PUSH$STACK;
02076  5                CALL MOVE$RB$RA;
02077  5                CALL ONE$VALUE$OPS(SIN);
C2078  5                CALL POP$STACK;
02079  5                CALL ONE$VALUE$OPS(COS);
C2080  5                CALL PUSH$STACK;
02081  5                IF RB$ZERO THEN
02082  5                     CALL ERROR('TZ');
02083  5                CALL TWO$VALUE$OPS(FDIV);
02084  5           END;
02085  4
02086  4     /* 73 SQUAREROOT */
02087  4           CALL ONE$VALUE$OPS(SQRT);
02088  4
02089  4     /* 74 TAB */
C2090  4           DO;
02091  4                CALL ROUND$CONV$BIN;
02092  5                IF (ARA := ARA - 1) >= PRINTBUFFER THEN
02093  5                     CALL DUMP$PRINT$BUFF;
02094  5                DO WHILE ARA > PRINTBUFFLENGTH;
02095  5                     ARA = ARA - PRINTBUFFLENGTH;
C2096  6                     END;
02097  5                PRINTBUFFER = ARA + PRINTBUFFERLOC;
02098  5           CALL POP$STACK;
C2099  5           END;
C2100  4
02101  4     /* 75 EXPONENTATION */
02102  4           CALL ONE$VALUE$OPS(EXP);
02103  4
02104  4     /* 76 FREE AREA IN FSA */
02105  4           DO;
02106  4                CALL PUSH$STACK;
02107  5                CALL FLOAT$ADDR(AVAILABLE(0));
02108  5           END;
C2109  4
C2110  4     /* 77  IRN  */
02111  4           DO;
02112  4                DECLARE SEED BASED SEEDLOC ADDRESS;
02113  5                SEED = ARA;
02114  5           END;
02115  4
02116  4     /* 78 LOG */
02117  4           CALL ONE$VALUE$OPS(LOG);
02118  4
02119  4     /* 79 POSITION OF PRINT BUFFER PTR */
C2120  4           DO;
02121  4                CALL PUSH$STACK;
02122  5                CALL FLOAT$ADDR(PRINTBUFFER - PRINTBUFFERLOC - 1);
02123  5           END;
02124  4
C2125  4     /* 80 INP */
02126  4           DO;
02127  4                INPUT: PROCEDURE(PORT) BYTE;
02128  6                     DECLARE
C2129  6                               PORT BYTE;
C2130  6                     GOTO PORTIN;
02131  6                END INPUT;
02132  5
02133  5                BRA(3) = INPUT(BRA);
02134  5                BRA(2) = 0;
C2135  5                ARA = 0;
02136  5                CALL CONV$TO$FP(RA);
```

```
02137  5              END;
02138  4
02139  4     /* 81 ASCII CONVERSION */
02140  4              DO;
02141  4                   DECLARE HOLD ADDRESS,
02142  5                           H    BASED    HOLD   BYTE;
02143  5                   IF (HOLD := ARA) = 0 OR H = 0 THEN
02144  5                        CALL ERROR('AC');
02145  5                   HOLD = HOLD + 1;
02146  5                   BRA(3) = H;
02147  5                   CALL STRING$FREE;
02148  5                   CALL FILL(RA,0,3);
02149  5                   CALL CONV$TO$FP(RA);
02150  5              END;
02151  4
02152  4     /* 82 CHR CONVERTS TO ASCII */
02153  4              DO;
02154  4                   DECLARE HOLD ADDRESS,
02155  5                           LOC BASED HOLD BYTE;
02156  5                   CALL CONV$TO$BIN$ADDR;
02157  5                   HOLD = GETSPACE(2);
02158  5                   LOC = 1;
02159  5                   LOC(1) = BRA;
02160  5                   ARA = HOLD;
02161  5                   CALL FLAGSTRINGADDR(TRUE);
02162  5              END;
02163  4
02164  4     /* 83 LEFT END OF STRING */
02165  4              CALL STRING$SEGMENT(0);
02166  4
02167  4     /* 84 LENGTH OF STRING */
02168  4              CALL FLOAT$ADDR(GET$STRING$LEN(ARA));
02169  4
02170  4     /* 85 MIDDLE OF STRING */
02171  4              CALL STRING$SEGMENT(2);
02172  4
02173  4     /* 86 RIGHT END OF STRING */
02174  4              CALL STRING$SEGMENT(1);
02175  4
02176  4     /* 87 CONVERSION TO STRING */
02177  4              DO;
02178  4                   CALL NUMERIC$OUT;
02179  5                   CALL MOVE(.PRINTWORKAREA,ARA :=
02180  5                        GETSPACE(PRINTWORKAREA + 1),PRINTWORKAREA + 1);
02181  5              END;
02182  4
02183  4     /* 88 VALUE  */
02184  4              CALL FP$INPUT(GET$STRING$LEN(ARA),ARA+1);
02185  4
02186  4     /* 89 COSH */
02187  4              CALL ONE$VALUE$OPS(COSH);
02188  4
02189  4     /* 90 SINH */
02190  4              CALL ONE$VALUE$OPS(SINH);
02191  4
02192  4     /* 91 RON  */
02193  4              CALL ROUND$CONV$BIN;
02194  4
02195  4     /* 92 CKO  */
02196  4              /* RA CONTAINS MAX NUMBER OF LABELS IN THE ON STATEMENT
02197  4                 RB CONTAINS SELECTED LABEL.
02198  4                 CHECK TO INSURE SELECTED LABEL EXISTS. IF NOT AN ERROR
02199  4                 HAS OCCURED */
02200  4              DO;
02201  4                   IF (BRB := BRB - 1) > BRA - 1 THEN
02202  5                        CALL ERROR('OI');
02203  5                   CALL POP$STACK;
02204  5                   ARA = (ARA * 3) + 1;
02205  5              END;
02206  4     /* 93 EXR  */
02207  4              CALL LOGICAL(3);
02208  4
02209  4
02210  4     /* 94 DEF  */
02211  4              DO;
02212  4                   CALL STEP$INS$CNT;
02213  5                   EOFBRANCH(GET$FILE$NUMBER) = TWOBYTEOPRAND;
02214  5                   CALL STEP$INS$CNT;
02215  5              END;
02216  4
02217  4
02218  4     /* 95  BOL */
02219  4              DO;
02220  4                   CURRENTLINE = ARA;
02221  5                   CALL POP$STACK;
02222  5              END;
02223  4
02224  4     /* 96  ADJ */
02225  4              ARA = ARA + MCD;
02226  4
02227  4              END;  /* END CASE */
02228  3         CALL STEP$INS$CNT;
02229  3         END;   /* OF DO FOREVER  */
02230  2
02231  2
02232  2
02233  2    END EXECUTE;
```

```
02234  1        /*
02235  1             ************************************************************
02236  1             *                                                          *
02237  1             ************************************************************
02238  1        */
02239  1
02240  1   MAINLINE:
02241  1        CALL CRLF;
02242  1        CALL INITIALIZE$EXECUTE;
02243  1   EOFEXIT:  /* ON END OF FILE OF CURRENT DISK FILE COME HERE */
02244  1   ERROR$EXIT:  /* REGROUP ON CONSOLE INPUT ERROR */
02245  1        CALL EXECUTE;
02246  1        CALL EXIT$INTERP;
02247  1   EOF
NO PROGRAM ERRORS
```

```
359.....END JOB BASIC1  .....END JOB BASIC1  .....END JOB BASIC1  .....END JOB BASIC1  .....
359.....END JOB BASIC1  .....END JOB BASIC1  .....END JOB BASIC1  .....END JOB BASIC1  .....
```

AD-A042 332 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
A MICROPROCESSOR IMPLEMENTATION OF EXTENDED BASIC.(U)
DEC 76 G E EUBANKS

UNCLASSIFIED NL

3 OF 3
ADA042332

END
DATE FILMED
8 - 77

LIST OF REFERENCES

1. Altair BASIC Reference Manual, MITS, Inc. 1975.

2. Dartmouth BASIC, Diewit Computation Center, Dartmouth College, Hanover, N.H. 1973

3. Digital Research, An Introduction to CP/M Features and Facilities, 1976.

4. Draft Proposed American National Standard Programming Language Minimal BASIC. X3J2/76-01 76-01-01. Technical Committee X3-J2-basic American National Standards Committee X3- Computers and Information Processing.

5. IBM, "IBM 5100 BASIC Reference Manual," May, 1976.

6. Intel Corporation, 8008 and 8080 PL/M Programming Manual, 1975.

7. Intel Corporation, INSITE Library Programs BB-38, BC-1, BC-2, and BC-4.

8. Intel Corporation, INSITE Library Program F-3.

9. Lientz, Bennet P., "A Comparative Evaluation of Versions of BASIC," Communications of the ACM, v. 19, n. 4, p 175-181, April 1976.

10. Lipp, Michael F., "The Language BASIC and Its Role In Time Sharing," Computers and Automation, October 1969.

11. Naval Postgraguate School Report NPS-53KD72 11A, ALGOL-E: An Experimental Approach to the Study of Programming Languages, by Gary A. Kildall, 7 January 1972.

12. Ogdin, Jerry L., "The Case Against ... BASIC," Datamation, v. 17, n. 17, p 34-41, 1 September 1971.

13. Sammet, Jean E., Programming Languages: History and Fundamentals, Prentice-Hall, 1969.

14. University of Toronto, Computer Systems Research Group Technical Report CSRG-2, "An Efficient LALR Parser Generator," by W. R. Lalonde, April 1971.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
2. Chairman, Code 52 Computer Science Department Naval Postgraduate School Monterey, California 93940	1
3. Assoc. Professor G. A. Kildall, Code 52Kd Computer Science Department Naval Postgraduate School Monterey, California 93940	1
4. LT Gordon E. Eubanks, Jr., USN USS George Washington (SSBN 598) (BLUE) FPO San Francisco, California 96601	1
5. Defense Documentation Center Cameron Station Alexandria, VA 22314	2